5-2018

Multi-Stop Routing Optimization: A Genetic Algorithm Approach

Abbas Hommadi
Utah State University

MULTI-STOP ROUTING OPTIMIZATION: A GENETIC ALGORITHM APPROACH

by

Abbas Hommadi

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

Vicki Allan, Ph.D. Major Professor	Nick Flann, Ph.D. Committee Member
Vladimir Kulyukin, Ph.D. Committee Member	Mark R. McLellan, Ph.D. Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2018

ABSTRACT

Multi-Stop Routing Optimization: A Genetic Algorithm Approach

by

Abbas Hommadi, Master of Science

Utah State University, 2018

Major Professor: Vicki Allan, Ph.D.
Department: Computer Science

The Traveling Salesman Problem (TSP) is one of the most important and attractive combinatorial optimization problems. There are many meta-heuristic algorithms that can solve this problem. In this paper, we use a Genetic Algorithm (GA) to solve it. GA has different operators selection, crossover, and mutation to address a solution to the problem. Sequential Constructive Crossover (SCX) and its modification Bidirectional Circular Constructive Crossover (BCSCX) are very efficient to solve TSP. Here, we propose a modification to these crossovers. The experimental results show that our proposed adjustment is superior to SCX and BCSCX as well as to other conventional crossovers (e.g. Order Crossover (OX), Cycle Crossover (CX), and Partially Mapped Crossover (PMX)) in term of solution quality and convergence speed. Furthermore, the GA solver (improved by applying inexpensive local search operators) can produce solutions with much better quality within reasonable computational time.

The Time-Dependent Traveling Salesman Problem (TDTSP) is an interesting problem and has an impact on real-life applications such as a delivery system. In this problem, time among destinations fluctuates during the day due to traffic, weather, accidents, or other events. Thus, it is important to recommend a tour that can save driver's time and

resources. In this research, we propose a Multi-Population Genetic Algorithm (MGA) where each population has different crossovers. We compare the proposed MGA against Single-Population Genetic Algorithm (SGA) in terms of tour time solution quality. Our finding is that MGA outperforms SGA. Our method is tested against real-world traffic data [1] where there are 200 different instances with different numbers of destinations (i.e. 60 different instances of 10 destinations, 60 different instances of 20 destinations, 60 different instances of 30 destinations, and 20 different instances of 50 destinations). For all tested instances, MGA is statistically superior on average by at least 10% (for instances with size less than 50) and 20% (for instances of size 50) better tour time solution compared to SGA with OX and SGA with PMX operators, and at least 4% better tour time compared to SGA with SCX operator.

(74 pages)

PUBLIC ABSTRACT

Multi-Stop Routing Optimization: A Genetic Algorithm Approach

Abbas Hommadi

In this research, we investigate and propose new operators to improve Genetic Algorithm's performance to solve the multi-stop routing problem. In a multi-stop route, a user starts at point x, visits all destinations exactly once, and then return to the same starting point. In this thesis, we are interested in two types of this problem. The first type is when the distance among destinations is fixed. In this case, it is called static traveling salesman problem. The second type is when the cost among destinations is affected by traffic congestion. Thus, the time among destinations changes during the day. In this case, it is called time-dependent traveling salesman problem. This research proposes new improvements on genetic algorithm to solve each of these two optimization problems.

First, the Travelling Salesman Problem (TSP) is one of the most important and attractive combinatorial optimization problems. There are many meta-heuristic algorithms that can solve this problem. In this paper, we use a Genetic Algorithm (GA) to solve it. GA uses different operators: selection, crossover, and mutation. Sequential Constructive Crossover (SCX) and Bidirectional Circular Constructive Crossover (BCSCX) are efficient to solve TSP. Here, we propose a modification to these crossovers. The experimental results show that our proposed adjustment is superior to SCX and BCSCX as well as to other conventional crossovers (e.g. Order Crossover (OX), Cycle Crossover (CX), and Partially Mapped Crossover (PMX)) in term of solution quality and convergence speed. Furthermore, the GA solver, that is improved by applying inexpensive local search operators, can produce solutions that have much better quality within reasonable computational time.

Second, the Time-Dependent Traveling Salesman Problem (TDTSP) is an interesting problem and has an impact on real-life applications such as a delivery system. In this problem, time among destinations fluctuates during the day due to traffic, weather, accidents, or other events. Thus, it is important to recommend a tour that can save driver's time and resources. In this research, we propose a Multi-Population Genetic Algorithm (MGA) where each population has different crossovers. We compare the proposed MGA against Single-Population Genetic Algorithm (SGA) in terms of tour time solution quality. Our finding is that MGA outperforms SGA. Our method is tested against real-world traffic data [1] where there are 200 different instances with different numbers of destinations. For all tested instances, MGA is superior on average by at least 10% (for instances with size less than 50) and 20% (for instances of size 50) better tour time solution compared to SGA with OX and SGA with PMX operators, and at least 4% better tour time compared to SGA with SCX operator.

To my parents Fadhil and Siham, to my wife Nada, and to my daughter Miral
without whom, I would never be who I am.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my major professor Dr. Vicki Allan for her valuable guidance, support, and patience during my graduate study. Without her persistence help and guide, this research would not be done.

Many thanks go to Dr. Nick Flann and Dr. Vladimir Kulyukin for serving on my committee. Their valuable discussion and assistance help this research to move forward.

A huge appreciation goes to the Higher Committee for Education Development in Iraq (HCED) for supporting my study at Utah State University. Also, my gratitude goes to the Computer Science Department at USU for allowing me to work as Graduate Teaching Assistant. This opportunity is not just helping me financially, but it also sharpens my problem solving and teaching skills.

Last but not least, I would express my sincere gratitude for my parents, brothers, and sisters who always support me. Special thanks to my lovely wife, Nada, and my adorable daughter, Miral, for your infinite love and unconditional sacrifices. Finally, I would like to thank all my awesome friends in Logan. I will never forget the great memories that we had together.

Abbas Hommadi

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER 1

INTRODUCTION

According to the urban mobility report from Texas A & M Transportation Institution, traffic congestion [2] can cause huge losses in money and time for the drivers. Figure 1.1 shows how much Salt Lake City (SLC) is affected by traffic congestion. In this Figure 1.1, a congestion cost is the value of the travel time delay and the wasted fuel consumption, where the fuel cost is the state average price per gallon of gasoline and diesel.

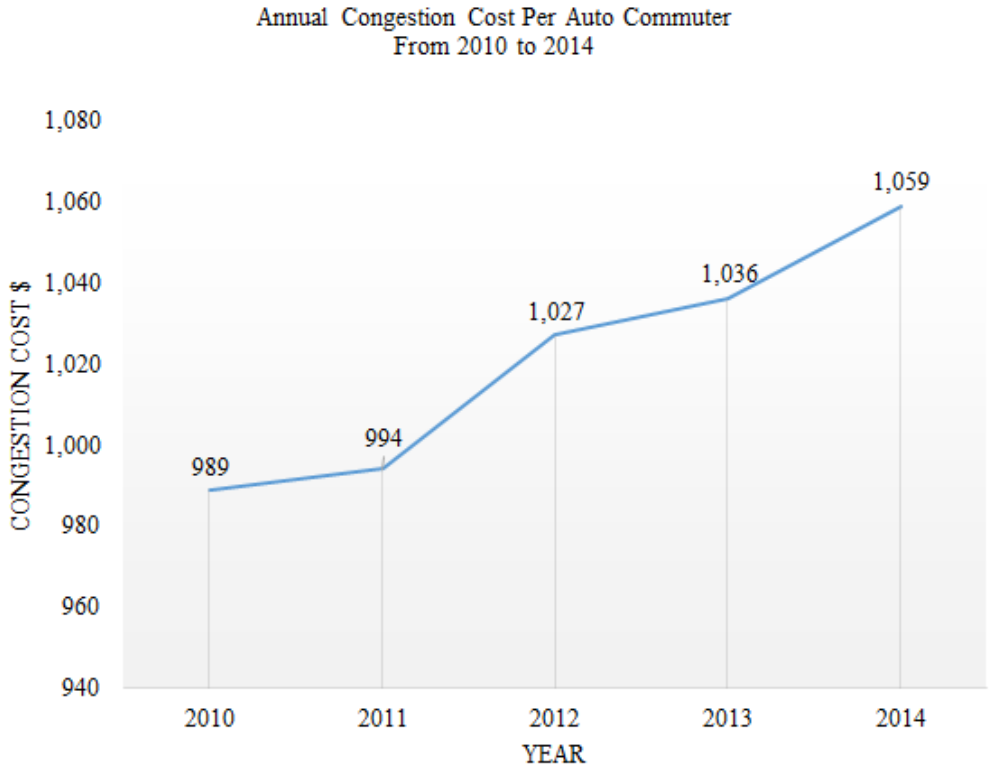


Fig. 1.1: Salt Lake City congestion cost per commuter.

Traffic congestion is a serious problem, and the community needs to use a better technology to reduce the costs while the population of a city increases. Sometimes, drivers

do not know about the historical and real-time traffic data. Thus, this leads drivers to drive on suboptimal roads. As a result, they lose money and time. Furthermore, congested traffic makes drivers impatient, which could make them more likely to cause accidents [3]. Moreover, using suboptimal roads definitely increases the city pollution. Therefore, using an intelligent method to plan routes is important. In this thesis, we are interested in two multi-stop routing problems: static traveling salesman problem, and time-dependent traveling salesman problem.

In the first part of this thesis, we study solving static Traveling Salesman Problem (TSP) using genetic algorithm. The TSP is one of the most important combinatorial optimization problems. It belongs to the class of NP-complete problems [4] [5]. Many real-world problems can be modeled as TSP such as, drilling of printed circuit boards, overhauling gas turbine engines, x-ray crystallography, computer wiring, the order-picking problem in warehouses, and vehicle routing problem [6]. Therefore, this problem attracts many researchers and many studies attempt a solution. TSP can be described as follows: There are n cities needed to be visited and a travel cost (e.g. distance, traffic time, money) matrix that represents the cost of traveling between each city and every other city. The goal is to find an optimal tour which starts at any city (say $City_{origin}$) and returns to the same city by visiting all other cities only once. Our solution focuses on the crossover and adding new operators to GA, where we developed and improved existing crossovers and applied different operators in an attempt to provide better solution quality and reasonable computational time for solving the TSP.

In the second part of this thesis, we consider a real-world optimization problem where time of the day plays an important role. Particularly, we are interested in the Time-Dependent Travelling Salesman Problem (TDTSP), an extended version of the static traveling salesman problem. In TDTSP, the cost between any two cities depends on the time of the day. TDTSP is the essence of many real-world problems such as, delivery systems where the driver wants to deliver goods starting from the depot, delivers the goods to each of the delivery places, and then comes back to the depot. In a real urban network, driving

time among deliveries fluctuates during the day and depends on the traffic of the roads. Although there are many studies done on the static traveling salesman problem, rare research has been done on TDTSP. In this research, we study Genetic Algorithm on TDTSP to develop a solution model for this optimization problem.

Each of the following two chapters is written to be self-contained paper.

CHAPTER 2

EFFICIENT GENETIC ALGORITHM FOR STATIC TSP

2.1 Abstract

The Travelling Salesman Problem (TSP) is one of the most important and attractive combinatorial optimization problems. There are many meta-heuristic algorithms that can solve this problem. In this paper, we use a Genetic Algorithm (GA) to solve it. GA uses different operators selection, crossover, and mutation to address a solution to the problem. Sequential Constructive Crossover (SCX) and its modification Bidirectional Circular Constructive Crossover (BCSCX) are very efficient to solve TSP. Here, we propose a modification to these crossovers. The experimental results show that our proposed adjustment is superior to SCX and BCSCX as well as to other conventional crossovers (e.g. Order Crossover (OX), Cycle Crossover (CX), and Partially Mapped Crossover (PMX)) in term of solution quality and convergence speed. Furthermore, the GA solver, that is improved by applying inexpensive local search operators, can produce solutions with much better quality within reasonable computational time.

2.2 Introduction

The Traveling Salesman Problem (TSP) is one of the most important combinatorial optimization problems. It belongs to the class of NP-complete problems [4] [5]. Many real-world problems can be modeled as TSP such as drilling of printed circuit boards, overhauling gas turbine engines, x-ray crystallography, computer wiring, the order-picking problem in warehouses, and vehicle routing problem [6]. Therefore, this problem attracts many researchers and many studies attempt a solution.

TSP can be described as follows: There are n cities needed to be visited and a travel cost (e.g. distance, traffic time, money) matrix that represents the cost of traveling between

each city and every other city. The goal is to find an optimal tour which starts at any city (say $City_{origin}$) and returns to the same city by visiting all other cities only once. In other words, given n cities with travelling cost matrix $D = [d_{i,j}]$ where $d_{i,j}$ is the cost of travelling from $city_i$ to $city_j$ $(i, j \in \{1, 2, ..., n\})$, the goal of the TSP is to find an optimal tour $T = (T_1, T_2, ..., T_n, T_1)$ that visits each city one and only one time and returns to the same starting city. Thus, the objective function can be represented as is given by 2.1.

$$f = \min \sum_{i=1}^{n-1} d_{T_i,T_{i+1}} + d_{T_n,T_1} \tag{2.1}$$

There are two types of static TSP. The first type is called symmetric TSP (STSP) where the cost between $City_i$ to $City_j$ is the same as the cost between $City_j$ to $City_i$. The second type is called asymmetric TSP (ATSP) where the cost is different if a path is travelled in the opposite direction. Thus, in this type $d_{ij} \neq d_{ji}$.

To solve this problem, one could try an exact exhaustive algorithm. Such an algorithm permutes all possible orderings for visiting n cities and picks the one that has minimum cost. This approach has $O(n!)$ complexity time where n is the number of cities. Although this method is simple and straightforward to implement, it is impractical and has exponential time. Therefore, approximate algorithms come in place. Approximate methods try to find a solution that has a quality close enough to the optimal solution with practical computation time. Therefore, many Artificial Intelligence (AI) techniques have been proposed to solve this problem. For example, some AI techniques are Genetic Algorithm (GA) [7], Simulated Annealing (SA) [8], Ant Colony Optimization (ACO) [9], Particle Swarm Optimization (PSO) [10], Tabu Search (TS) [11], and Neural Network (NN) [12]. Because of the strength and effectiveness of GA compared with other methods in solving combinatorial optimization problems [13], we have chosen to solve TSP using genetic algorithms.

Our solution focuses on the crossover and adding new operators to GA, where we developed and improved existing crossovers and applied different operators in an attempt to provide better solution quality and reasonable computational time for solving the TSP.

2.2.1 A Brief Introduction to Genetic Algorithms

The genetic algorithm is one of the best evolutionary algorithms used to solve the TSP problem [14]. It was first introduced by John Henry Holland [7]. The genetic algorithm is motivated by natural selection and genetics to find an approximated solution for a given problem. It exploits and explores the search space by the evolving process. The process of simple genetic algorithm shown in Figure 2.1.

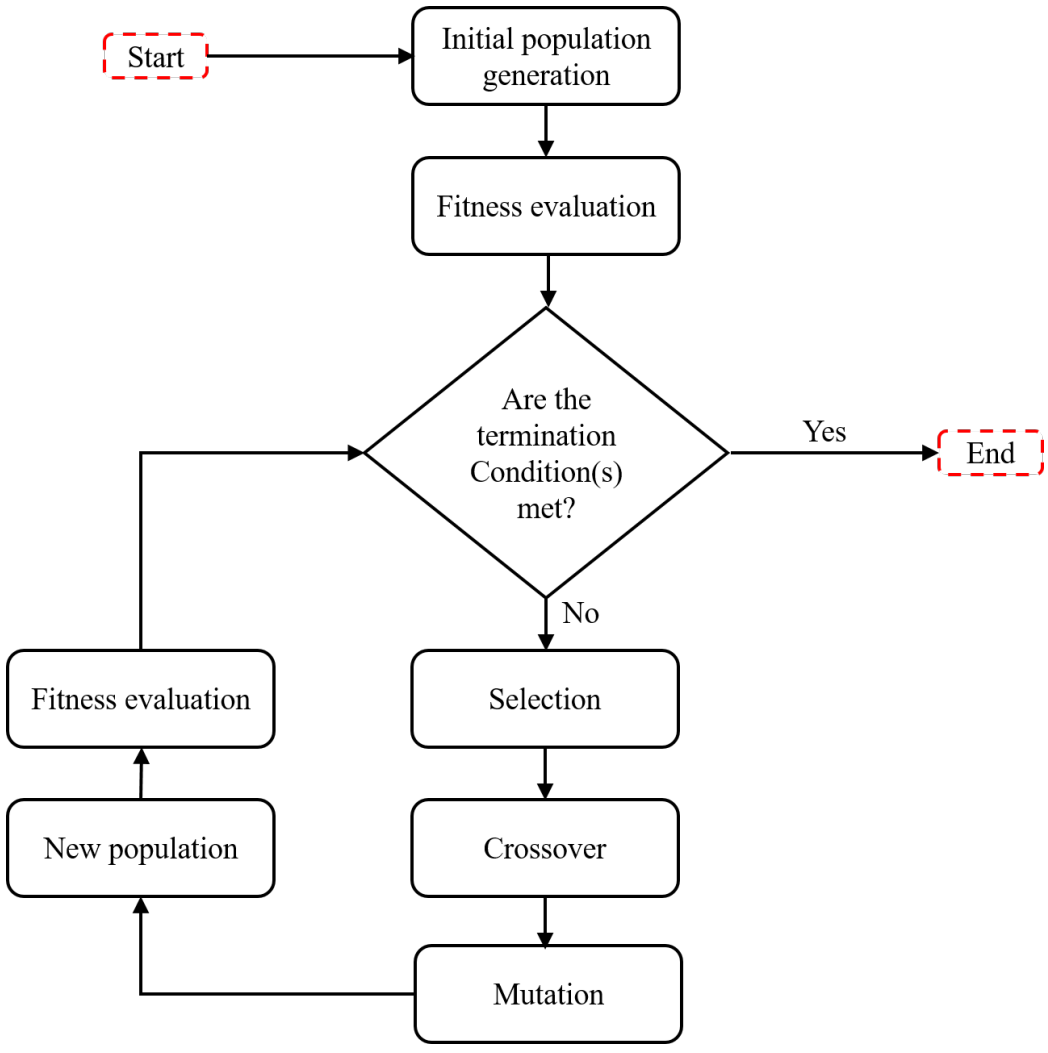


Fig. 2.1: Genetic algorithm flow chart.

- Initial Population: This is the first phase where an initial population is generated

either randomly (which is recommended for most problems) or the initial population seeded in a way that the optimal solution is more likely to be found [15].

- Fitness Evaluation: This is a function which used to evaluate the goodness for each individual (e.g. a possible solution) in the population.
- Termination Condition: There are some common termination conditions that could be used to stop the evolving process: a solution that satisfies the minimum criteria is achieved, a given number of generation is reached, computation time is reached, no progress is observed, or a combination of these conditions [15].
- Selection: This process is used to select individuals from the current generation that could be used for the next generation.
- Crossover: This process used to mix or mate the selected parent to produce offspring that could be inserted to the next generation.
- Mutation: This step is used to exploit the search space by making very small changes of an individual.

2.3 Literature Review

Many studies have been done on solving TSP with GA using different operators. In this section, an overview and summary of these studies are provided.

The research of Noraini and Geraghty [16] focuses on finding which best selection method could be used to solve TSP. They compare different and well-known selection strategies: tournament selection, proportional roulette wheel selection, and rank base roulette wheel selection. This comparison study is based on solution quality and a number of generations to reach the best solution. Their results show that the tournament selection is more suitable than the other selection methods for small size (less than 50 cities) tour in terms of good solutions (i.e. near-optimal solution), and the number of generations to converge. Additionally, for a large tour size, the rank based roulette wheel is more effective to give good quality of solution. The reason for this behavior is that tournament selection could

suffer from premature convergence by being influenced by super individuals in the population. On the other hand, rank based roulette wheel selection overcomes the premature convergence by uniformly scale the ranks of individuals across the population. Thus, rank based roulette wheel selection does not get influenced by super individuals in the population, but this selection is more computationally expensive in contrast to the tournament selection which is much cheaper.

Paul et al. [17] analyze different population initialization methods for TSP based on different criteria such as computation time, convergence speed, and error rate. Their study covered random initialization, nearest neighbor initialization, gene bank, sorted population, selective initialization, and ordered distance vector. They made their experiments on TSPLIB [18] and found out that the nearest neighbor method is better in term of computation time and convergence speed, but it limits the exploration of search space. Therefore, it suffers from premature convergence. On the other hand, ordered distance vector outperforms the other initialization methods.

Abdoun et al. [19] investigated crossover operators to solve TSP. They considered six different crossovers: Uniform Crossover Operator, the Cycle Crossover, the Partially-Mapped Crossover, the Uniform Partially-Mapped Crossover, the Non-Wrapping Ordered Crossover and the Ordered Crossover (OX). They conclude that OX gives the best solution in terms of solution quality.

Grefenstette et al. [20] introduced one of the first heuristic crossovers to solve the TSP which is Greedy Crossover (GX). The core idea of GX is when it selects a place to go on the tour, it will consider all its four neighbors in the two parents for next place to be visited. Then, it chooses the nearest neighbor to be visited next. This process continues until the tour gets completed. Some researchers came up with several versions of this crossover in an attempt to make improvements. These versions differ mainly in solving the special cases in this process. The special case here is when all four neighbors are already visited. [20] suggest picking the next place randomly, while [21–24] consider the second nearest place then the third and so on. Ismkhan and Zamanifar [25] suggested an improved greedy crossover (IGX).

In this method, instead of probing all four neighbors of the current place, it only probes the ones that have not been visited yet by using a doubly linked list as auxiliary storage. They showed that IGX behaves better in term of solution accuracy and time complexity for solving TSP.

Ahmed [14] suggested a new crossover especially designed for TSP called Sequential Constructive Crossover (SCX). The concept of this crossover is to obtain new offspring from the parents by taking advantage of existing good edges that are present in the structure of parents. Furthermore, it has the ability to introduce new good edges in the offspring. The researcher did a comparison study between the suggested one (SCX) and two other crossovers which are edge recombination crossovers [26] and generalized N-points crossover [27]. They found SCX gives a better solution for the TSP in terms of solution quality. SCX has one weakness which is that it always starts at first gene in the first parent when building new offspring and that leads to influence all the remaining genes while our method overcomes this weakness by random start when constructing offspring.

Wang et al. [28] propose an improved genetic algorithm to solve TSP. They added a new operator called the untwist operator that helps to untie the knots in the route and that will make the convergence speed faster. The following Figure 2.2 shows the untwisting operator.

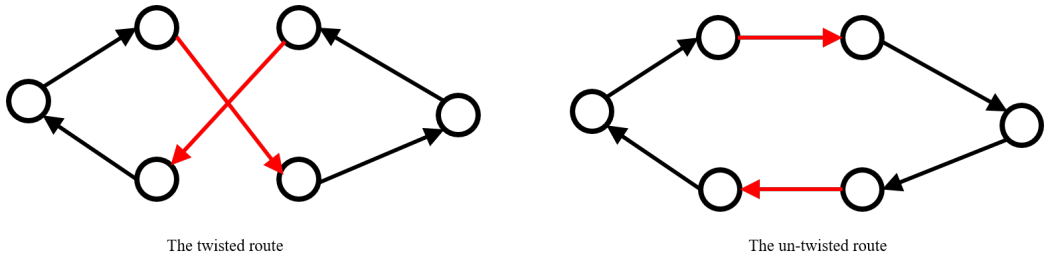


Fig. 2.2: The un-twisting operator

They made their experiments with 50, 100, and 150 cites, and they found out that this untwisting operator helps to speed up convergence and gives a good and reasonable solution for TSP.

Abdoun et al. [29] made a comparative study of different mutation operators of genetic algorithms for solving TSP to analyze which one performs best. Their comparison analysis based on six different mutations: twors mutation, center inverse mutation, reverse sequence mutation, throas mutation, thrors mutation, and partial shuffle mutation. Twors mutation swaps two randomly selected genes. Center inverse mutation splits genes to two sections and then reverses the order of genes in each section. Reverse sequence mutation selects a random consecutive subset then reverses the order of genes in the subset. Throas mutation is the same as reverse sequence mutation but here the subset length is always three. Thrors mutation takes three inconsecutive genes that are randomly selected then reverses their order in offspring. Partial shuffle mutation takes subset of genes and then randomly shuffle this subset. In their experiments, they found that reverse sequence mutation and partially shuffle mutation gives the best solution for TSP.

2.4 Proposed Method

In this paper, we use several crossovers and introduce some modifications on these crossovers to solve static TSP. Furthermore, we propose to use a local search for GA that helps to get better tour quality. This method is shown as Algorithm 1.

The best individuals get transferred from the current generation to the next one without any modification. This set is termed the **elitist individuals**. A random initial population is used to get random feasible solutions of the problem. The stopping criteria is based on reaching a maximum number of generations. Tournament selection with size 2 is used in this study to select individuals from the current population to be parents of the next generation. Because of the ease of implementation and its efficiency, this selection strategy is commonly used in GA [16] [30]. This approach works by selecting n individuals randomly and then selecting the best of n. When n is two, this approach is called binary tournament, and it is commonly used [16]. Although this selection method keeps the level of diversity, it slows down the convergence of the GA. However, it has several advantages which are efficient time complexity, low vulnerability to dominant fittest individuals, and no prior

requirements for sorting or scaling of fitness function [16] [31] [30].

Algorithm 1: Genetic Algorithm

Data: TSP

Result: BestRoute

```
Population ← InitializePopulation(TSP)
Evaluate(Population)
NumGen ← 1
while NumGen ≤ MaxGens do
    Offspring ← Select(Population, PopulationSize − ElitistIndividualsSize)
    Crossover(Offspring)
    Mutate(Offspring)
    OptimizeOperator(Offspring)
    Evaluate(Offspring)
    Population ← ElitistIndividuals + Offspring
    Update(ElitistIndividuals)
    NumGen ← NumGen + 1
end
BestRoute ← GetFittest(Population)
```

For the mutation step, swap mutation with probability P_m is used. Use of a mutation operator helps to escape local optima and keeps the diversity of a population. This step exploits the search space [29]. This mutation works by randomly selecting two cities and exchanging the position of these cities as shown in Figure 2.3.

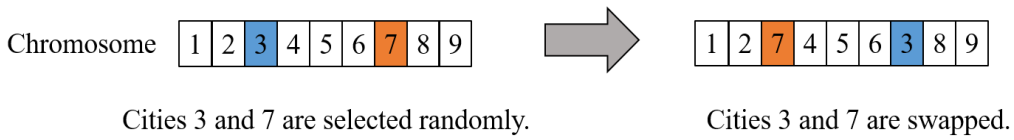


Fig. 2.3: Swap mutation.

2.4.1 Crossover Operators

The crossover operator is one of the basic operators of genetic algorithms. This operator is applied according to a probability P_{cx} on the selected chromosomes. It plays an important role in the performance of the genetic algorithm. The idea behind this operator is to combine two solutions in the search space to produce offspring that have better features and could survive next generations. In TSP problem, this operator could serve as a local search for the problem by building a new improved solution(s) using the knowledge presents in both parents [32] [33]. Although conventional crossover operators are simple and very straightforward to implement as shown in Figure 2.4, they cannot be applied to TSP problem without modifications. The reason is that TSP does not allow for missing and duplicated cities in a tour. Thus, a more intelligent crossover needs to be applied for TSP. Therefore, researchers introduced and developed crossovers specially designed for TSP. Any crossover could be applied to the TSP if it respects the problem constraints and the gene encoding that is used to represent a solution.

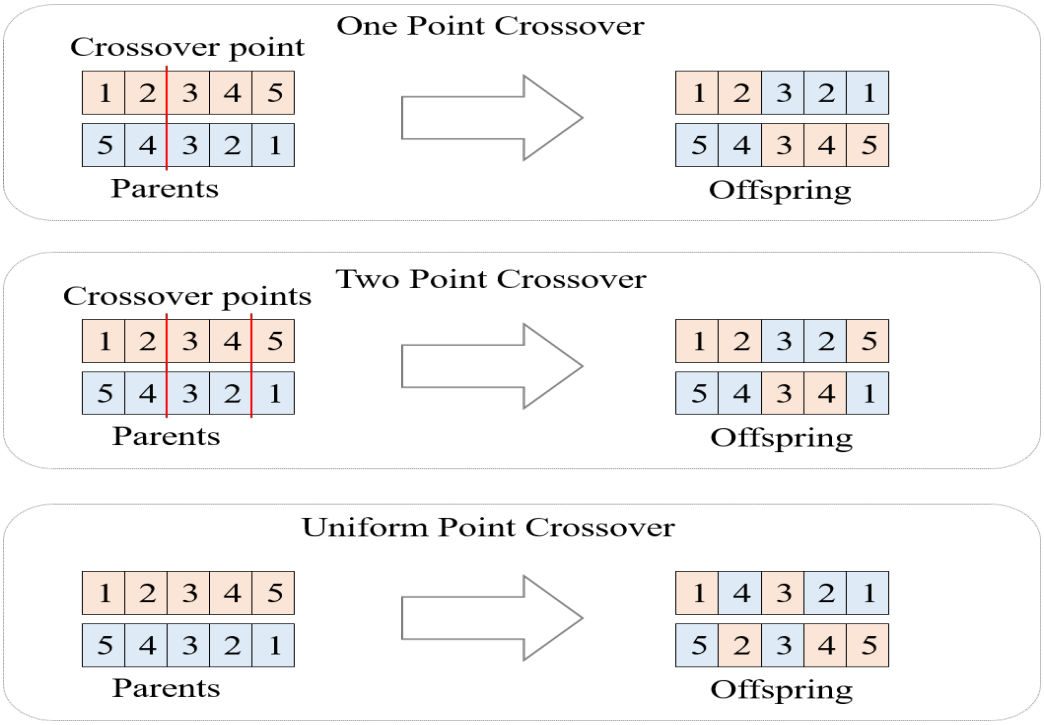


Fig. 2.4: Conventional crossover operators examples.

In this section, crossovers adapted to TSP will be described and illustrated by detailed examples. These crossovers are Order Crossover (OX) [34], Cycle Crossover (CX) [33], Partially Mapped Crossover (PMX) [35], Sequential Constructive Crossover (SCX) [14], Enhanced Sequential Constructive Crossover (ESCX) [36], and Bidirectional Circular Sequential Constructive Crossover (BCSCX) [37].

2.4.1.1 Order Crossover (OX)

According to [34] this crossover works as following:

Step 1: Randomly selects a sub-list of consecutive genes from the first parent.

Step 2: Produce the first offspring by copying the selected sub-gene list from the first parent into the corresponding positions of the first offspring.

Step 3: Start on the right edge of selected sub-list in parent 2, copy genes from parent 2 to the right edge of sub-list of offspring 1. Wrap around and skip duplicated genes if necessary.

Step 4: Produce the second offspring by switching parents roles.

Figure 2.5 demonstrates OX procedure.

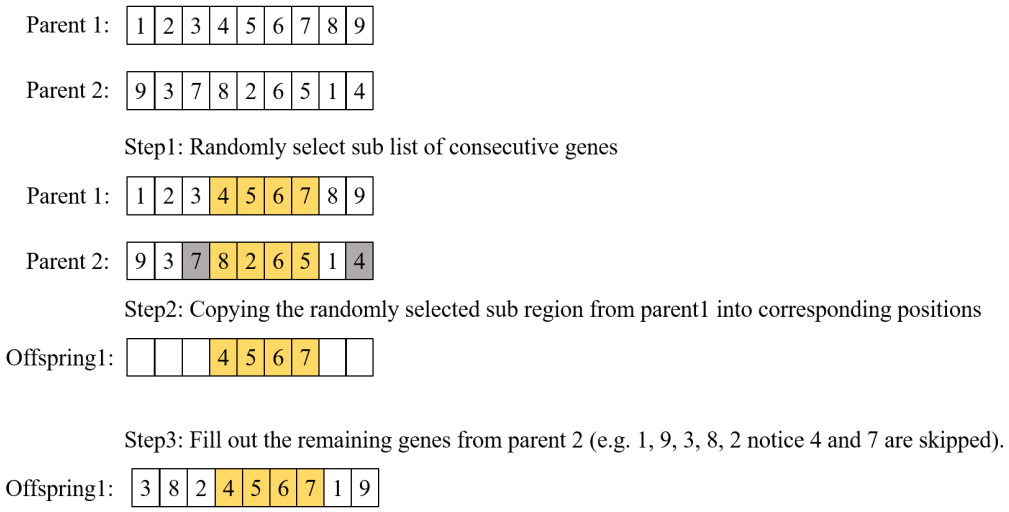


Fig. 2.5: Order crossover example.

2.4.1.2 Cycle Crossover (CX)

According to [33] the cycle crossover is designed to reserve information about gene position in parents and it works as follows:

Step 1: Find cycles

(a) Begin with first g_i in parent 1.

(b) Look at g_i the same position in parent 2 (i.e. gene at index i in parent 2). Call this gene g_j.

(c) Search for g_j in parent 1.

(d) Add g_j to the cycle.

(e) Repeat step b to d until you reach g_i of parent 1.

Step 2: Construct the two offspring by selecting cycles alternatively from both parents.

The following example in Figure 2.6 and Figure 2.7 illustrates the process.

2.4.1.3 Partially Mapped Crossover (PMX)

As stated in [35], partially mapped crossover reserves a slice of genes from one parent and keeps the order from the other one. The following example in Figure 2.8 demonstrates the procedure of this crossover.

2.4.1.4 Sequential Constructive Crossover (SCX)

The sequential constructive crossover is introduced by Ahmed [14] and the following steps explain the method:

Step 1: Begin with first gene p in parent 1.

Step 2: Sequentially search for next neighbor gene of p in both parents. If there is no legitimate gene (i.e. not yet visited) found, then find the next gene in legitimate genes list (i.e. list of unvisited genes). Let α is the gene obtained from parent 1 and β is the gene obtained from parent 2.

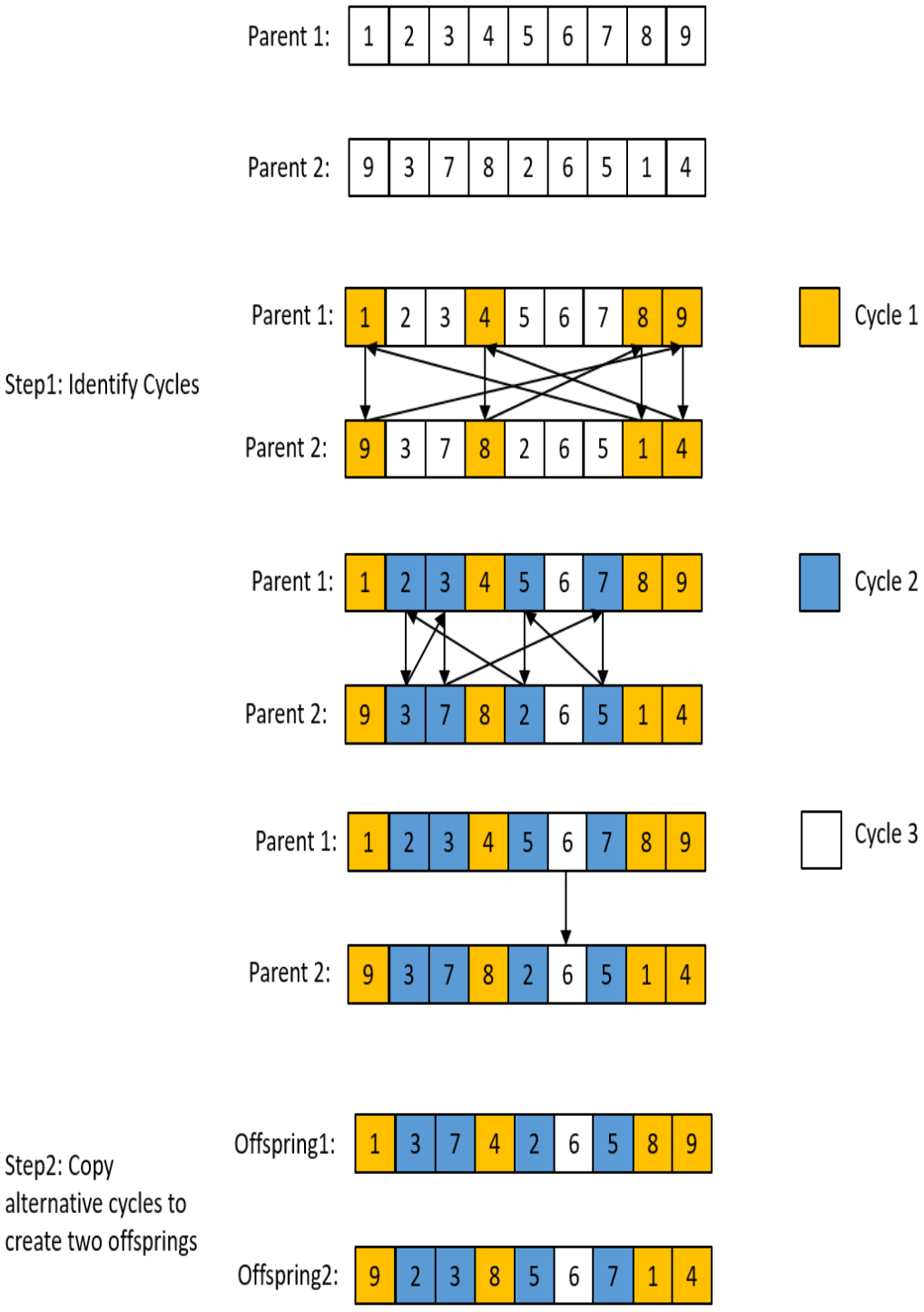


Fig. 2.6: Cycle crossover example.

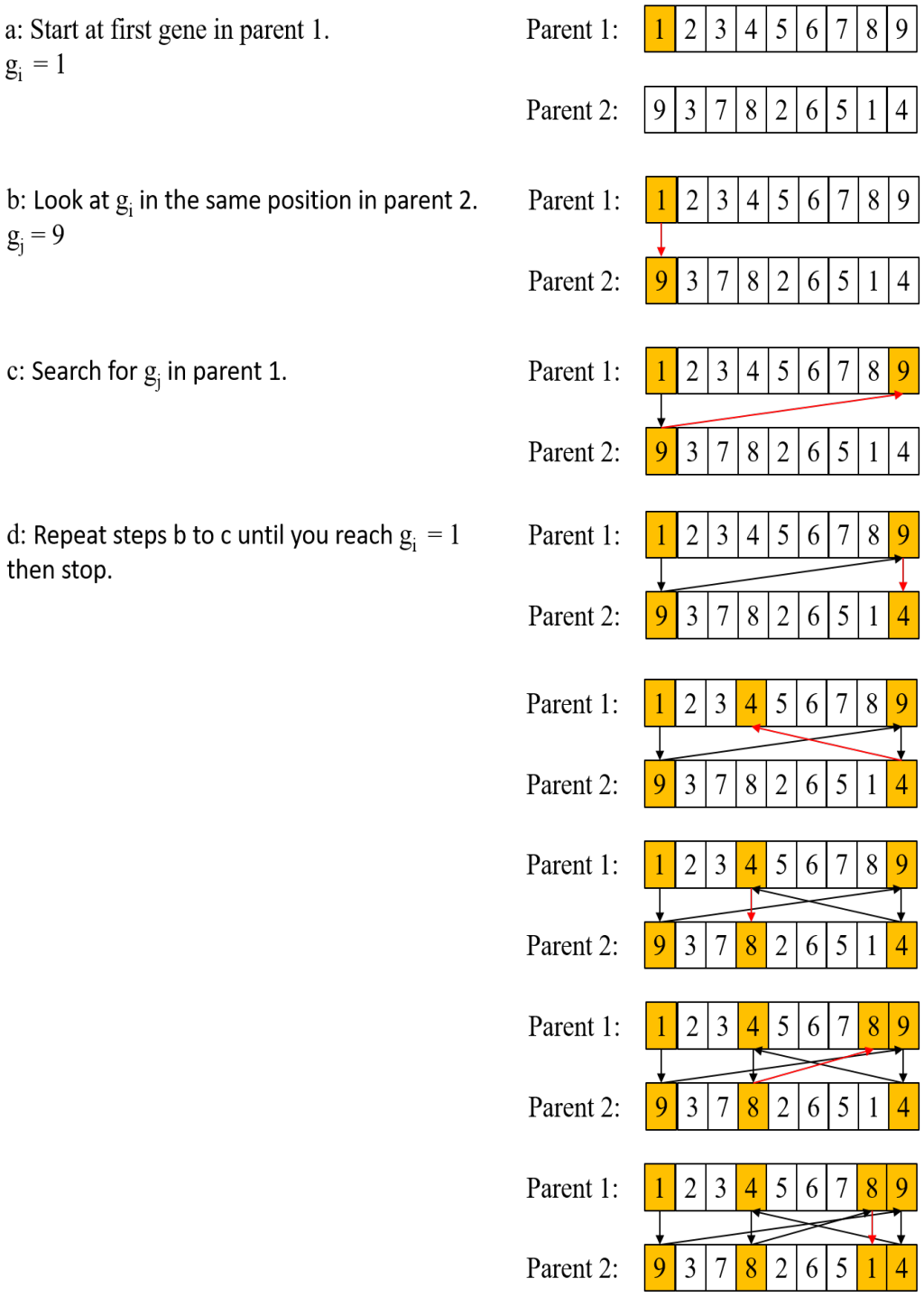


Fig. 2.7: Cycle constructing in CX example.

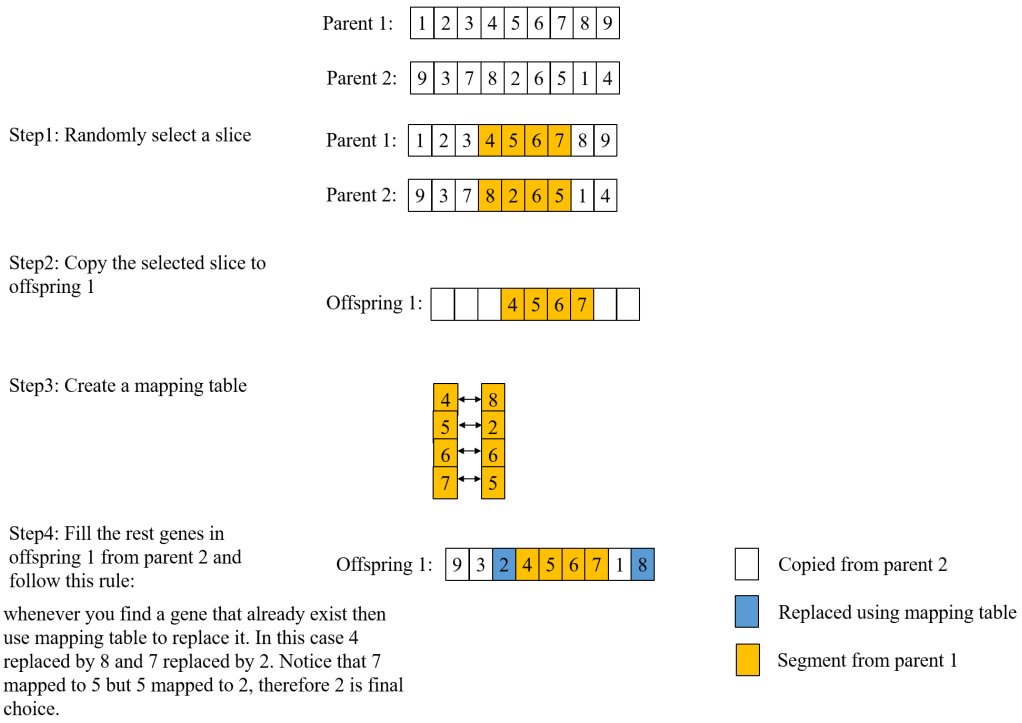


Fig. 2.8: Partially mapped crossover example.

Step 3: If $Cost_{p,\alpha} < Cost_{p,\beta}$ then choose geneα otherwise choose geneβ as next gene and append it to the partially constructed offspring.

Step 4: If the offspring complete, then stop, otherwise, set p to the chosen gene and continue on step 2.

See the following Figure 2.9 for more details.

2.4.1.5 Enhanced Sequential Constructive Crossover (ESCX)

Enhanced sequential constructive crossover is introduced by Hachemi and Alanzi [36] in an attempt to enhance SCX. The crossover is inspired by A* algorithm. It takes into its consideration not only the cost from the current gene to next possible gene but also it considers the minimum cost between the next possible gene and all the remaining unvisited genes. By this way, it constructs offspring heuristically. Refer to the following example 2.10 for more details.

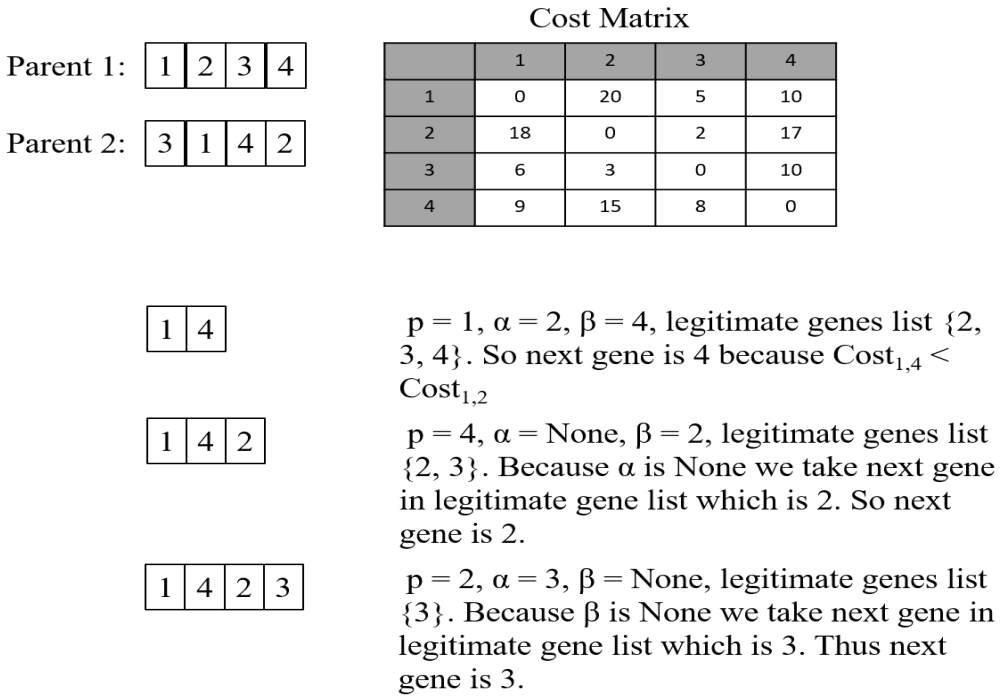


Fig. 2.9: Sequentially constructive crossover example.

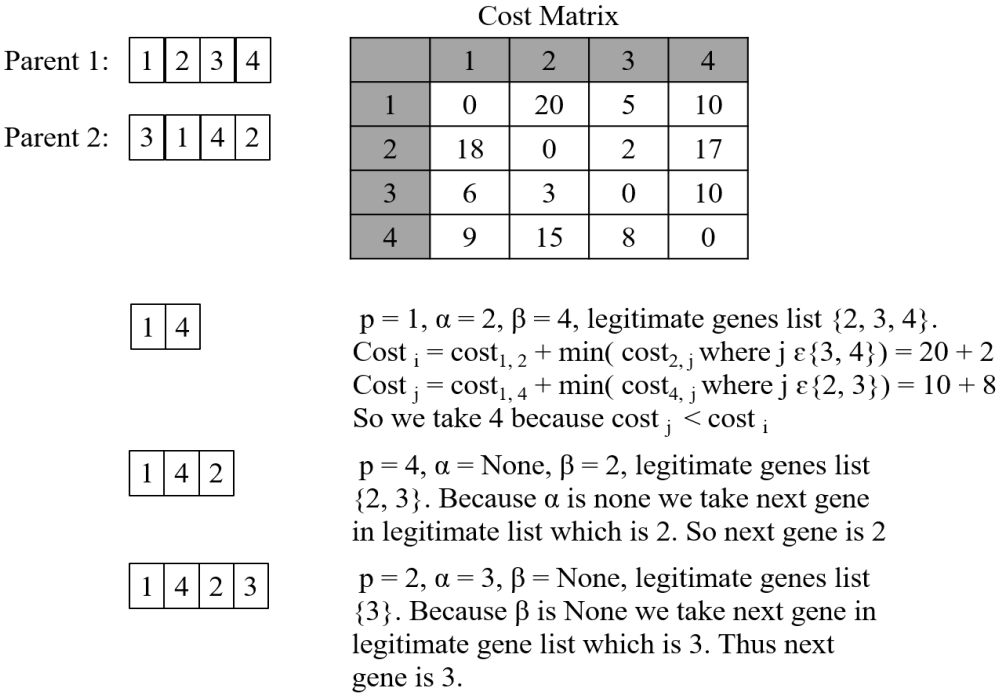


Fig. 2.10: Enhanced sequential constructive crossover example.

2.4.1.6 Bidirectional Circular Sequential Constructive Crossover (BCSCX)

Bidirectional circular sequential constructive crossover is introduced by [37] by modifying the SCX. It works just like SCX except for these two modifications:

1. Search for next neighbor that occurs in both directions (left and right) in both parents. Thus, in this approach, four genes as neighbors will be considered.

2. During searching for the next neighbor gene, if it reaches to the end or to the beginning of the genes list in any of the parents, it will wrap around.

See Figure 2.11 for further explanations.

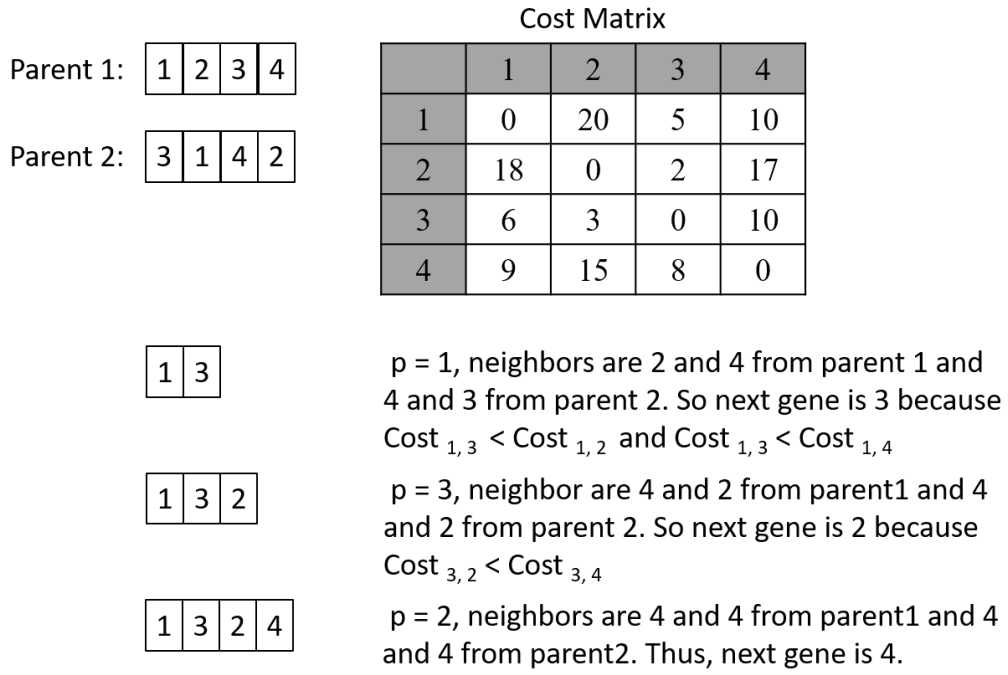


Fig. 2.11: Bidirectional circular sequential constructive crossover example.

2.4.1.7 Our Modified Crossover

In the original SCX and BCSCX, constructing offspring always picks the first gene in the first parent then adds this picked gene to be the first gene in the offspring. This policy results in chromosomes with an unchanged first gene. As result, this policy guides the

greedy search that is followed by SCX and BCSCX for constructing the rest of the genes in the offspring. Thus, to overcome this weakness, we propose picking the first gene to start with randomly. In other words, we pick a random gene g_i in parent 1 and copy it to the corresponding location into the offspring. Then we construct the remaining genes (starts from g_{i+1} to g_{i-1}) by following the same procedure in SCX and BCSCX. We named this crossover Random Start SCX (RSSCX) and Random Start BCSCX (RSBCSCX).

2.4.2 Local Optimization Operator

In addition to the three basic GA operators (e.g. selection, crossover, and mutation), sometimes there is a need to add another operator to increase the convergence speed of GA and enhance the solution quality. In this regard, we use a non-uniform local search operator [38] to improve our GA solver for TSP. This operator works as follows.

Three different cities (i, j, k) are selected randomly. Then we try all 3! possible combinations of i, j, k in the tour to find the best tour. The three cities are exchanged in their positions without any effect of other positions of unselected cities in the tour. The following Algorithm 2 describes this operator.

Algorithm 2: Non-Uniform Local Search Operator

Data: Individual: a chromosome to be modified
Result: BestIndividual: a modified chromosome

```
1  City_i, City_j, City_k ← chooseRandomCity(Individual)
2  AllPossibleCombinations ← Permute(City_i, City_j, City_k)
3  AllPossibleIndividuals ← Modify(Individual, AllPossibleCombinations)
4  BestIndividual ← Individual
5  foreach PossibleIndividual ∈ AllPossibleIndividuals do
6      Evaluate(PossibleIndividual)
7      if PossibleIndividual is better than BestIndividual then
8          BestIndividual ← PossibleIndividual
9      end
10 end
11 return BestIndividual
```

2.5 Experiments and Results

2.5.1 Experiment Setup

This section will focus on the quality of solutions found in our experiments using our modified GA crossovers and operators discussed in this study to obtain an optimal tour for static TSP. The algorithms are coded in Python 3. The performance of GA is tested on different TSPLib benchmarks [18]. For all experiments, we used random population initialization. We used tournament selection with tournament size set to 2. Swap mutation is applied in this experimental study. The objective of this experimental study is to investigate the performance of GA using different crossovers and furthermore improve the GA solver by applying the local search optimization operator in terms of solution quality.

One of the challenging things in building a practical GA is choosing appropriate values for parameters such as population size, crossover probability(P_c), and mutation probability(P_m). For this experiment, the guidelines of De Jong and Spears [39] has been followed which recommend starting relatively high P_c and relatively low P_m, and the population size is selected approximately 10 times larger than the number of cities in a problem. The maximum number of generations is chosen as stopping criteria for the GA and it is set earlier in the program.

2.5.2 Experiment Results

2.5.2.1 Crossovers' Solution Quality and Performance

2.5.2.1.1 Symmetric TSP To study the effectiveness of our proposed crossover compared to other crossover operators in GA, we solve each TSPLIB instance using GA with each one of the crossover operators. These crossovers are order crossover(OX), cycle crossover (CX), partially mapped crossover(PMX), sequential constructive crossover(SCX), enhanced SCX(ESCX), bi-directional circular SCX(BCSCX), as well as our proposed ones which are : random start SCX(RSSCX), random start BCSCX(RSBCSCX). The test is repeated 20

times for each crossover then we compute the average of best solutions for 20 runs per TSPLIB instance per crossover. Finally, we take the percentage change of the average best solutions over the optimal solutions using this Equation 2.2.

$$PercentChange = \frac{(SolutionValue - OptimalSolutionValue)}{OptimalSolution} \times 100 \qquad (2.2)$$

In Figure 2.12, we show crossovers comparison based on the quality of solutions.

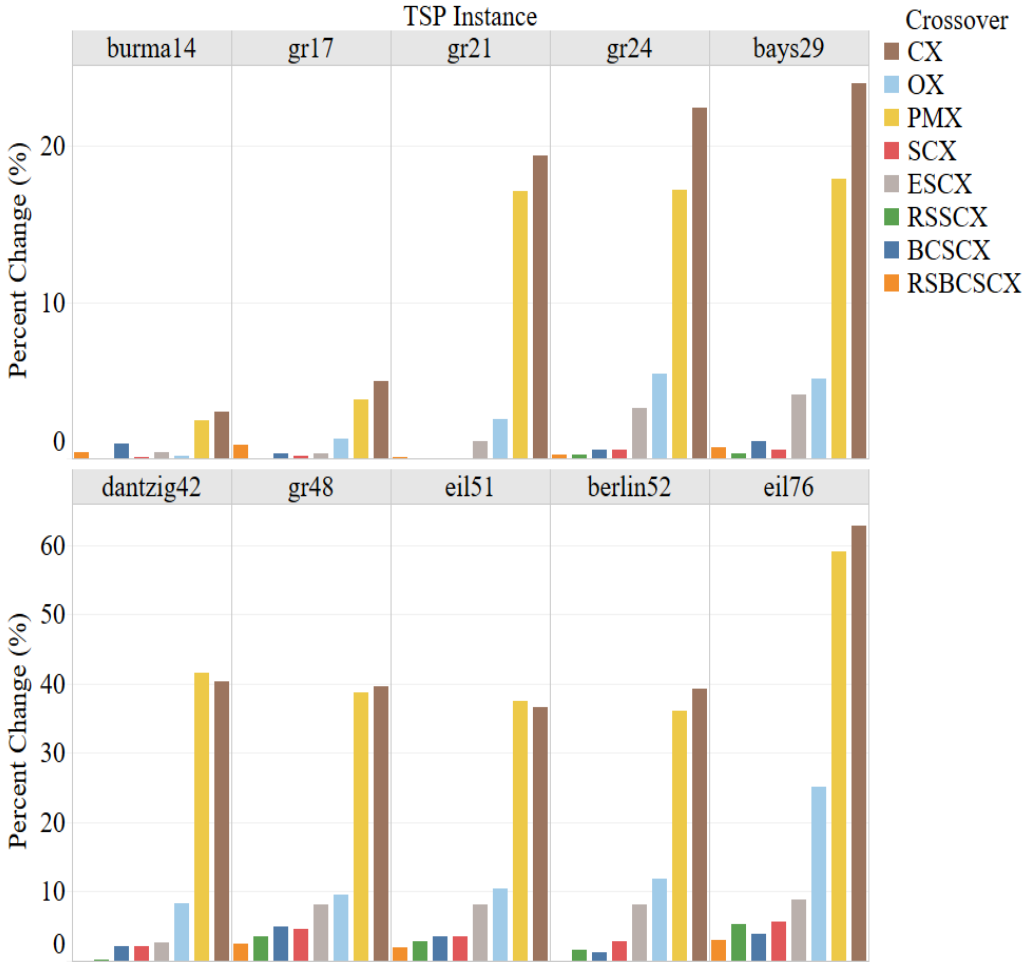


Fig. 2.12: Crossover comparison based on solution quality for symmetric TSP.

As it is shown in this Figure 2.12, the solutions obtained by applying constructive crossover (SCX, ESCX, BCSCX, RSSCX, and RSBCSCX) are better than the solutions

obtained from traditional crossover operators (OX, CX, and PMX). The reason for this behavior is that traditional crossover operators' process is based on reordering genes in a chromosome in some way without considering any knowledge about the tour itself to be built. In comparison, constructive crossover operators use some of the available information (i.e. distance to the neighbor cities) to construct a better tour. The other observation can be seen in Figure 2.12 is that the proposed RSSCX and RSBCSCX are superior to their original operators SCX and BCSCX in terms of the solution quality for all tested TSP instances. This is an important indication that a random start when building offspring as used in RSSCX and RSBCSCX makes the crossover achieve better chromosomes and is not blindly biased to the first city in parent 1 chromosome as in SCX and BCSCX. Moreover, it seems that SCX and RSBCSCX have about the same solution quality for small instances (less than 30 cities) except in some cases where SCX is better (e.g. burma14, gr17, and bays29) but for larger problem (greater than 30 cities) RSBCSCX outperforms RSSCX for all instances (e.g. dantzig42, gr48, eil51, berlin52, eil76). Each benchmark is named to indicate city size.

The performance graphs in Figures 2.13, 2.14, 2.15, 2.16, 2.17, 2.18, 2.19, 2.20, 2.21, and 2.22 show the convergence over generations of GA using different crossovers when it applied to solve different TSPLIB instances (e.g. burma14, gr17, gr21, gr24, bays29, dantzig42, gr48, eil51, berlin52, and eil76). As we can see in these figures, the cost (e.g. distance) gets reduced in subsequent generations and eventually converges at specific generation. For example, Figure 2.13 shows the convergence of solving burma14 instance. We can see that RSSCX and RSBCSCX converge to nearly optimal cost after at most 20 generation. On the other hand CX, OX, and PMX take much longer to converge. That is a valuable indication that RSSCX and RSBCSCX take reasonable time to come up with near optimal solutions.

In Figures 2.23, 2.24, and 2.25, we plot the solution found by applying RSSCX as crossover in each TSPLIB instance.

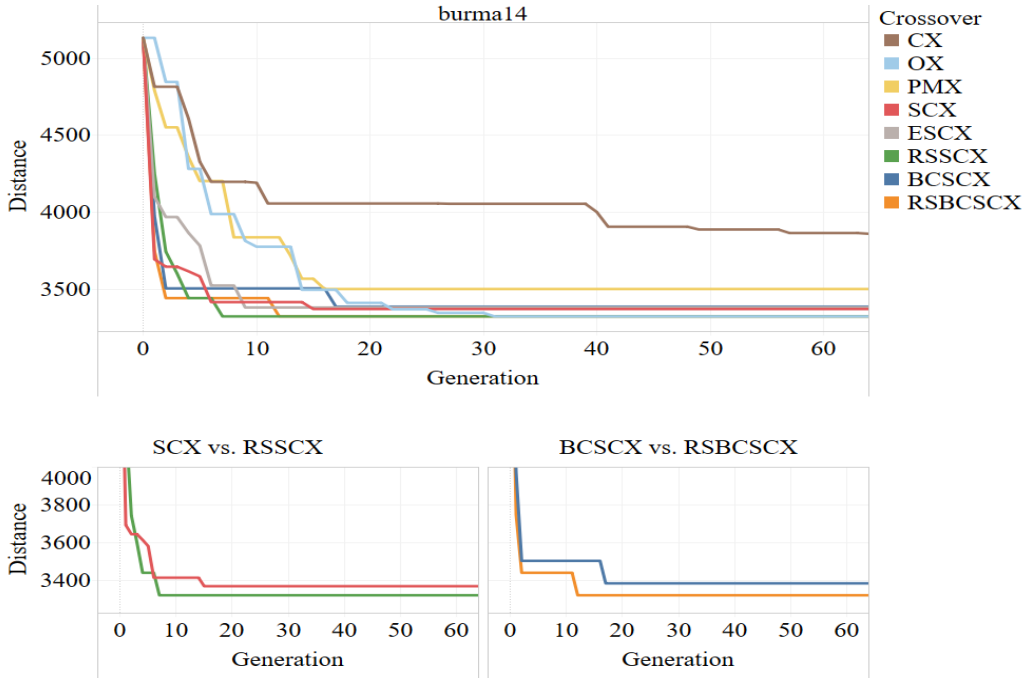


Fig. 2.13: Performance graph for burma14 instance showing convergence over generations

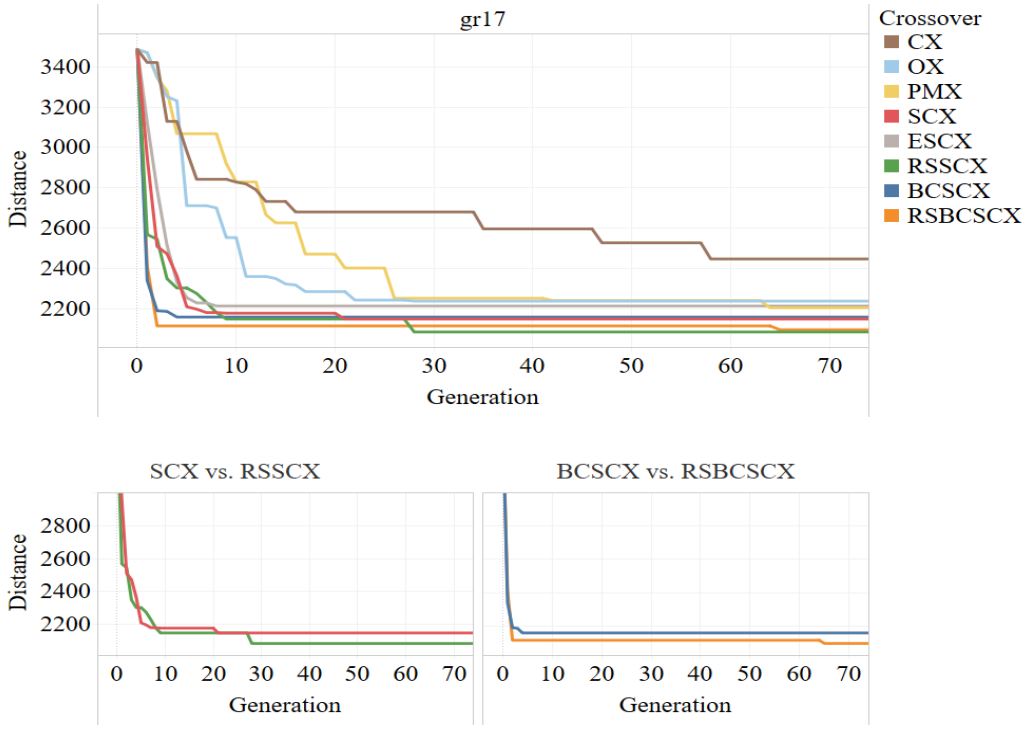


Fig. 2.14: Performance graph for gr17 instance showing convergence over generations

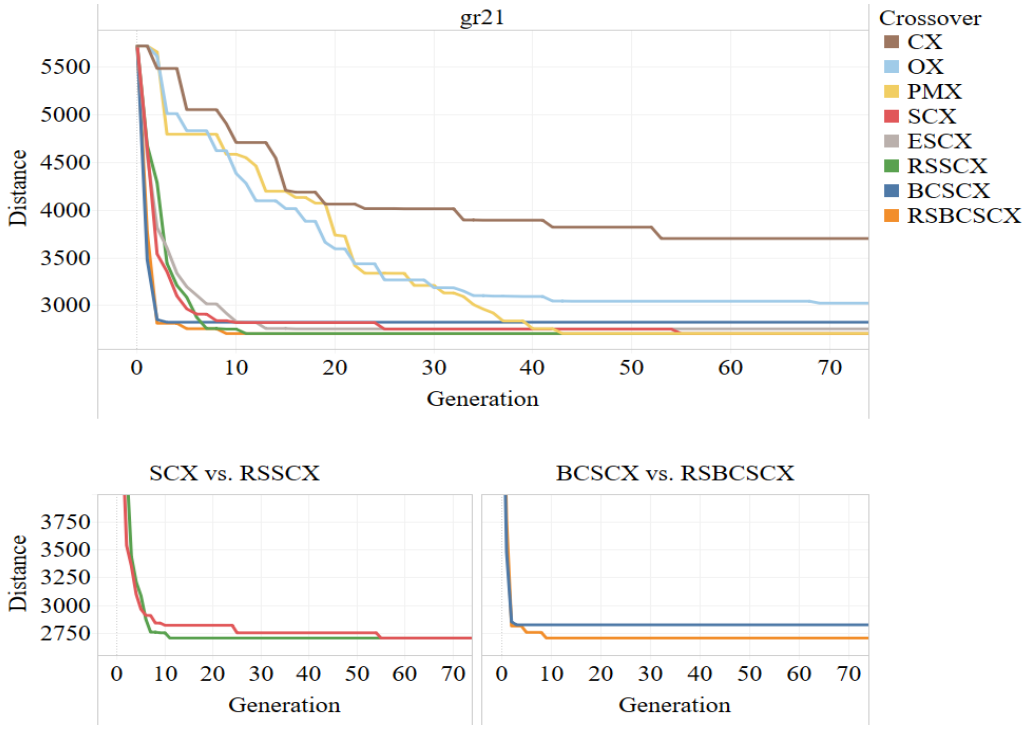


Fig. 2.15: Performance graph for gr21 instance showing convergence over generations

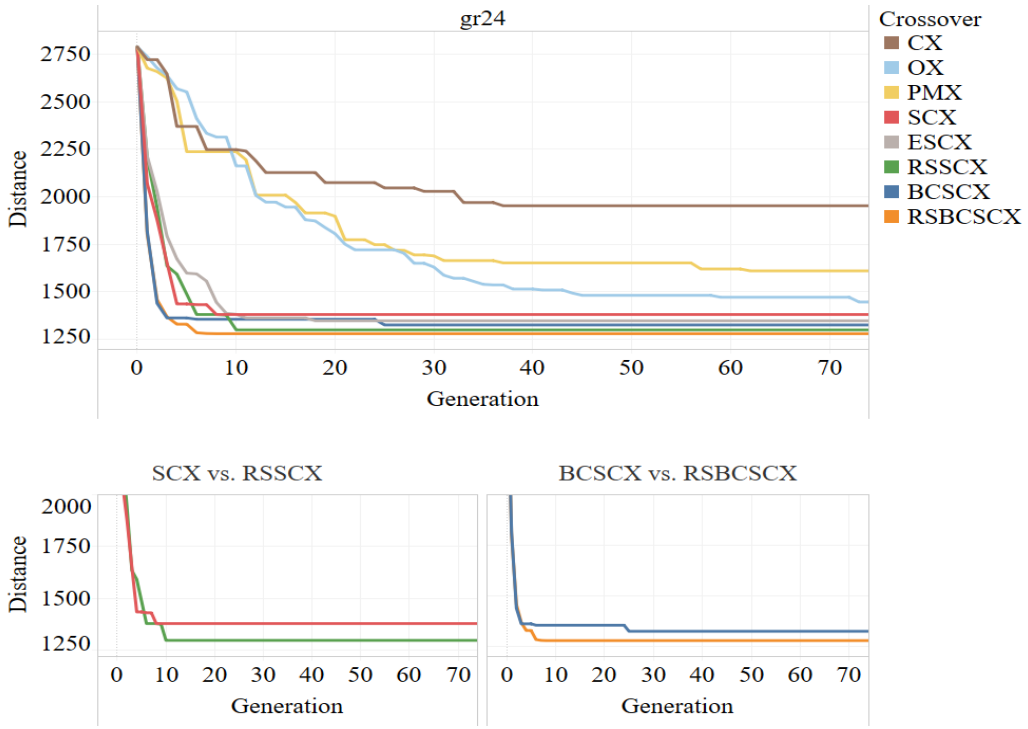


Fig. 2.16: Performance graph for gr24 instance showing convergence over generations

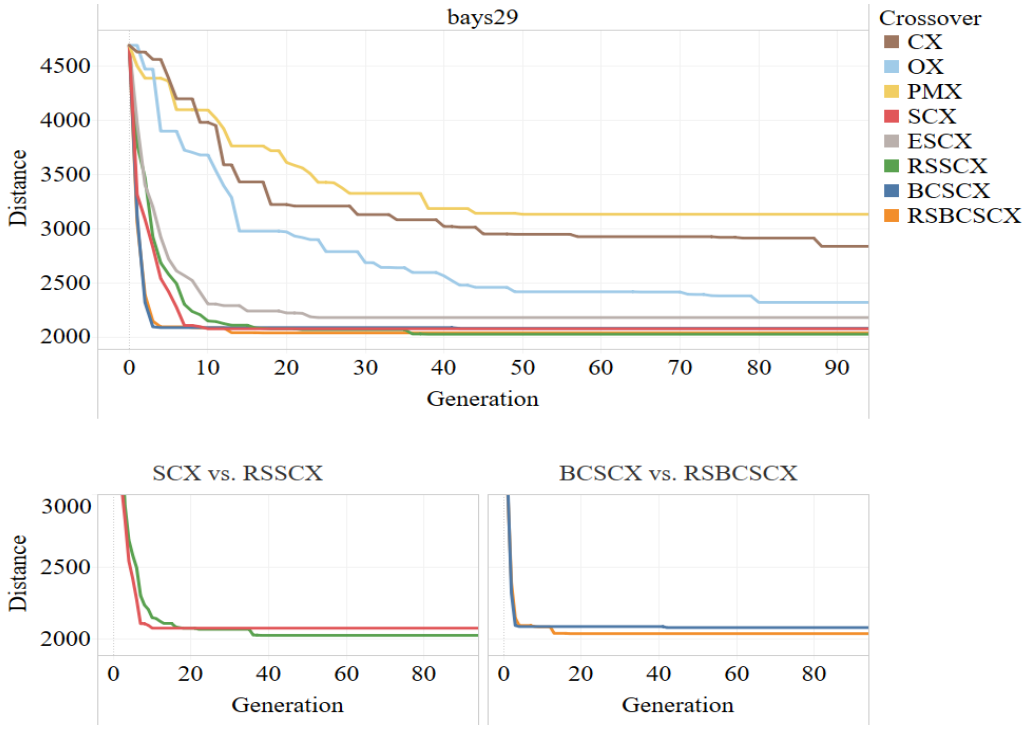


Fig. 2.17: Performance graph for bays29 instance showing convergence over generations

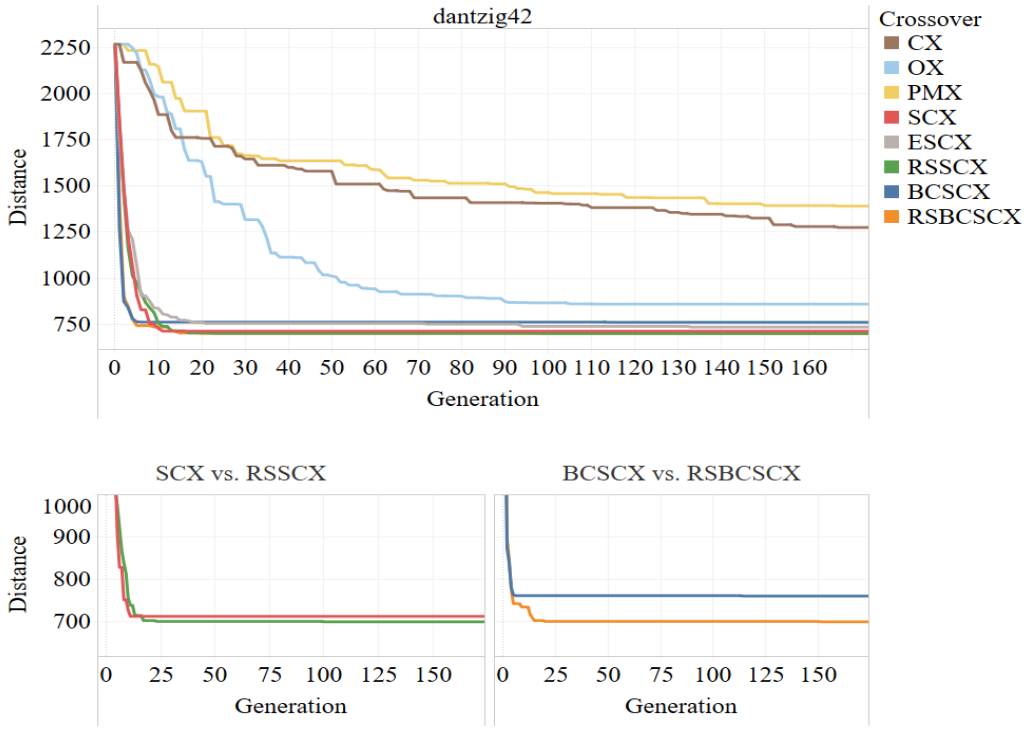


Fig. 2.18: Performance graph for dantzig42 instance showing convergence over generations

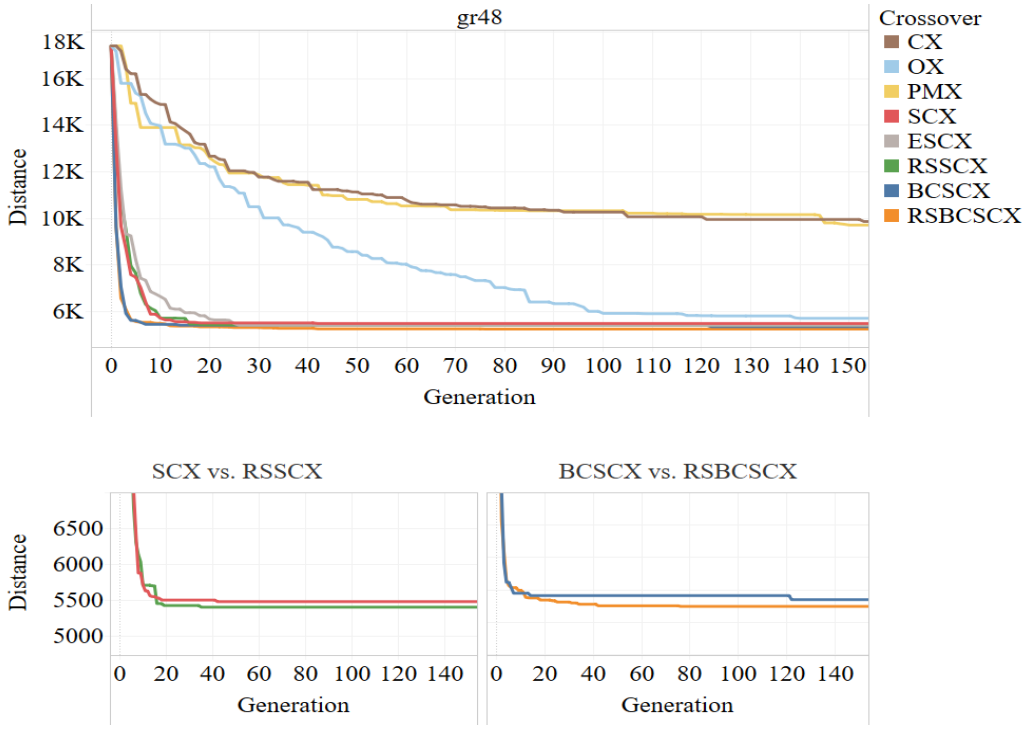


Fig. 2.19: Performance graph for gr48 instance showing convergence over generations

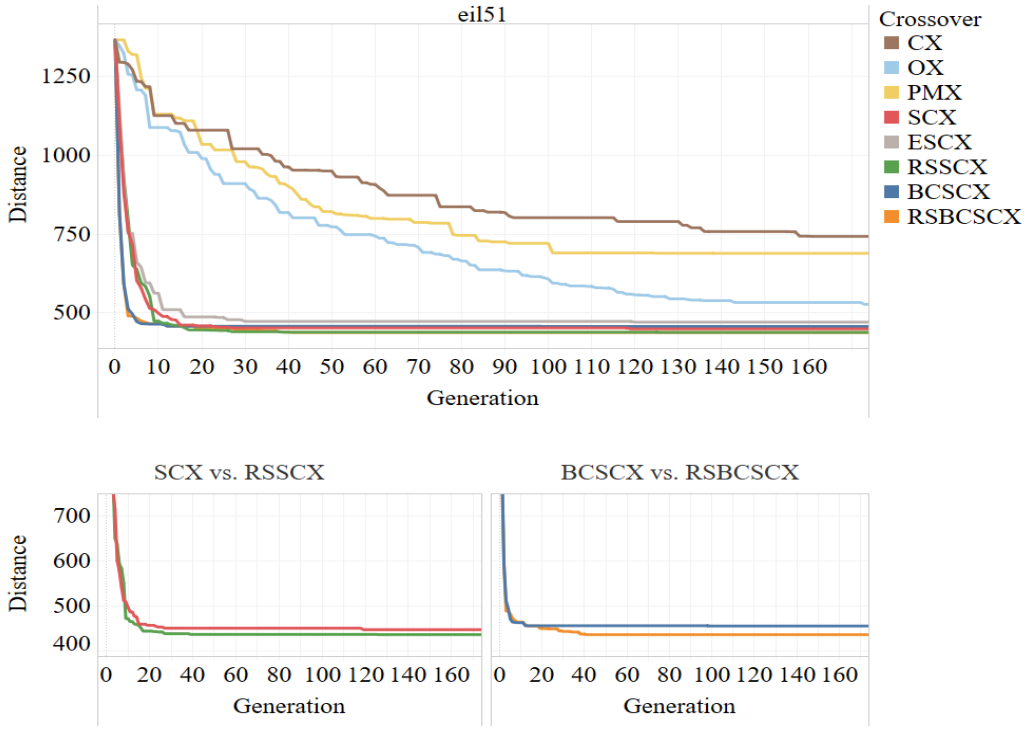


Fig. 2.20: Performance graph for eil51 instance showing convergence over generations

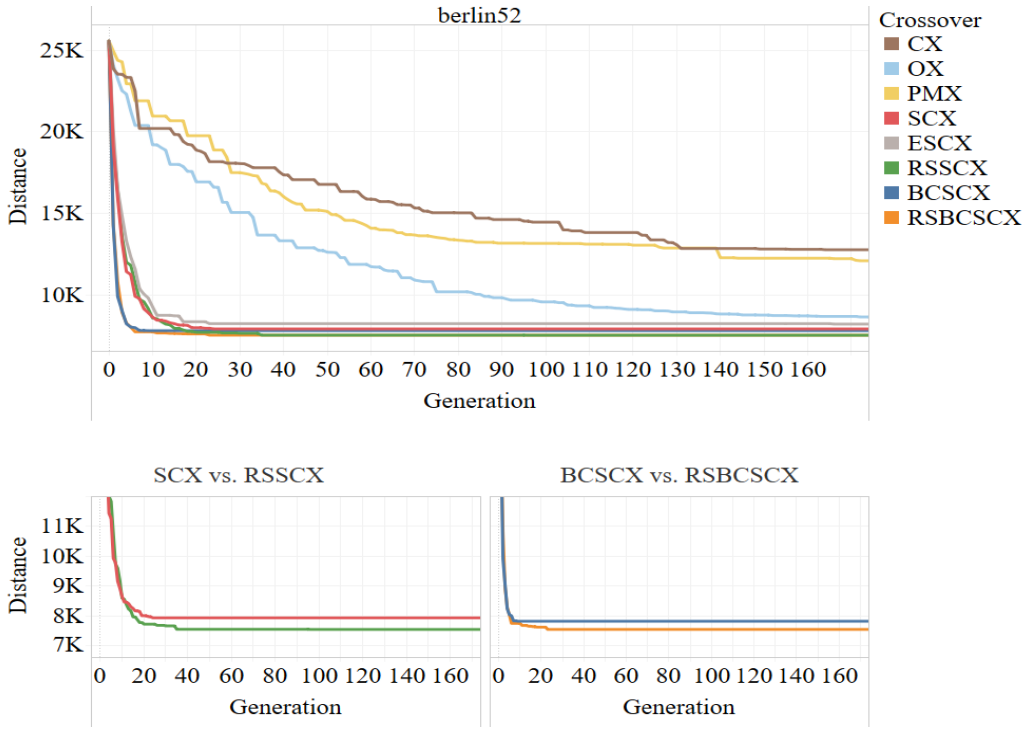


Fig. 2.21: Performance graph for berlin52 instance showing convergence over generations

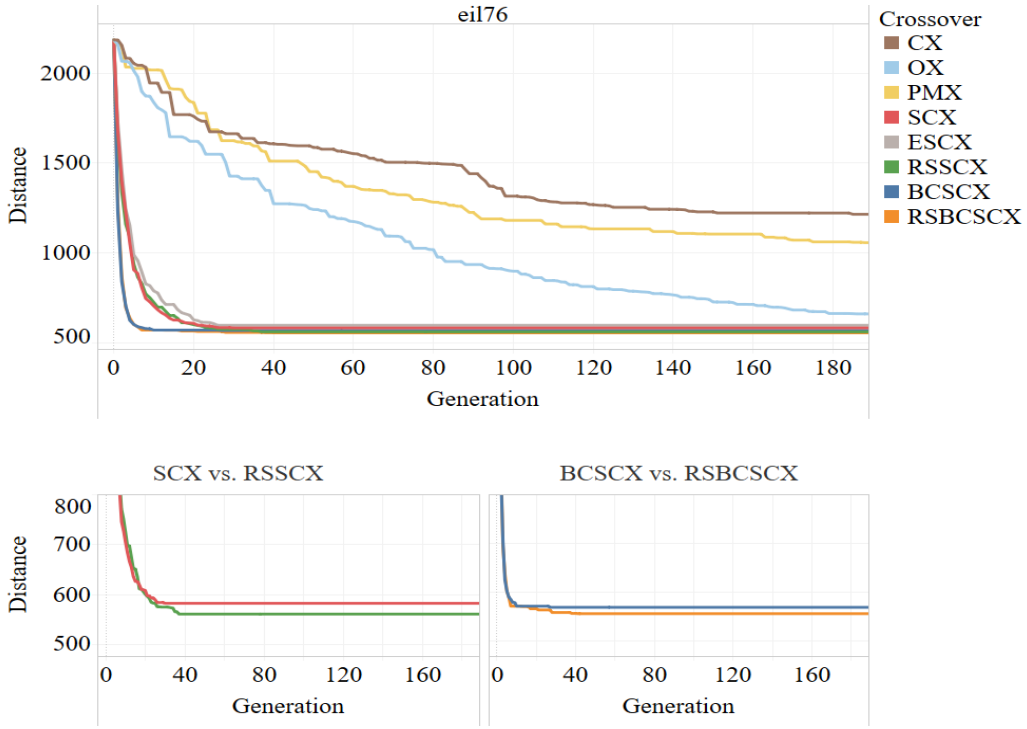


Fig. 2.22: Performance graph for eil76 instance showing convergence over generations

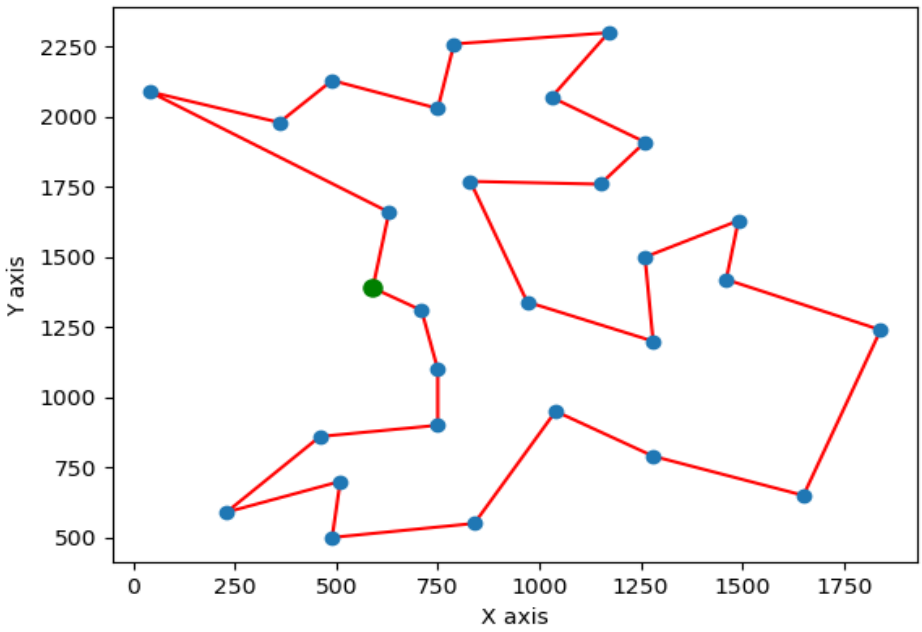


Fig. 2.23: Optimal soultion of bays29 instance when using RSSCX as crossover. The total distance is 2020.

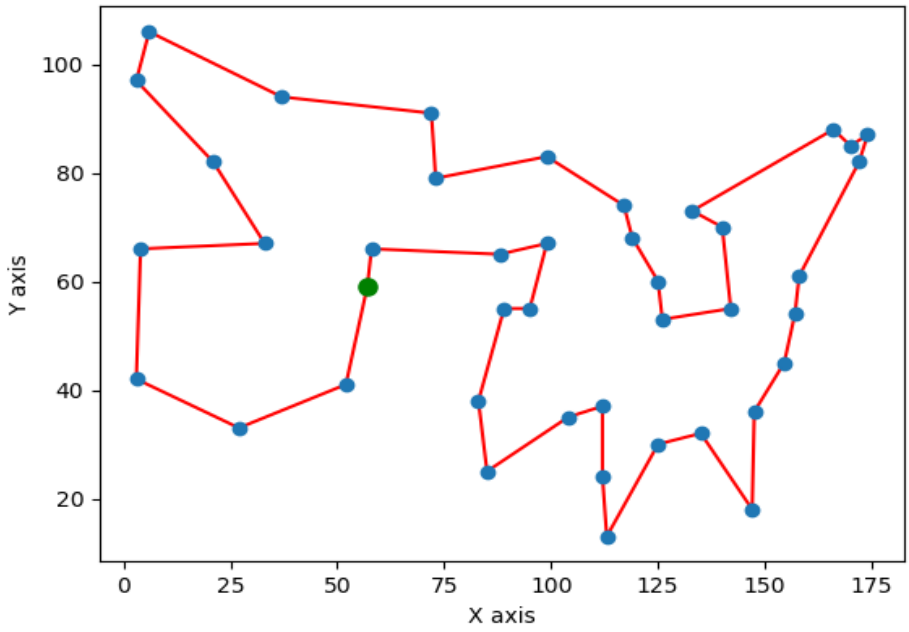


Fig. 2.24: Optimal soultion of dantzig instance when using RSSCX as crossover. The total distance is 699.

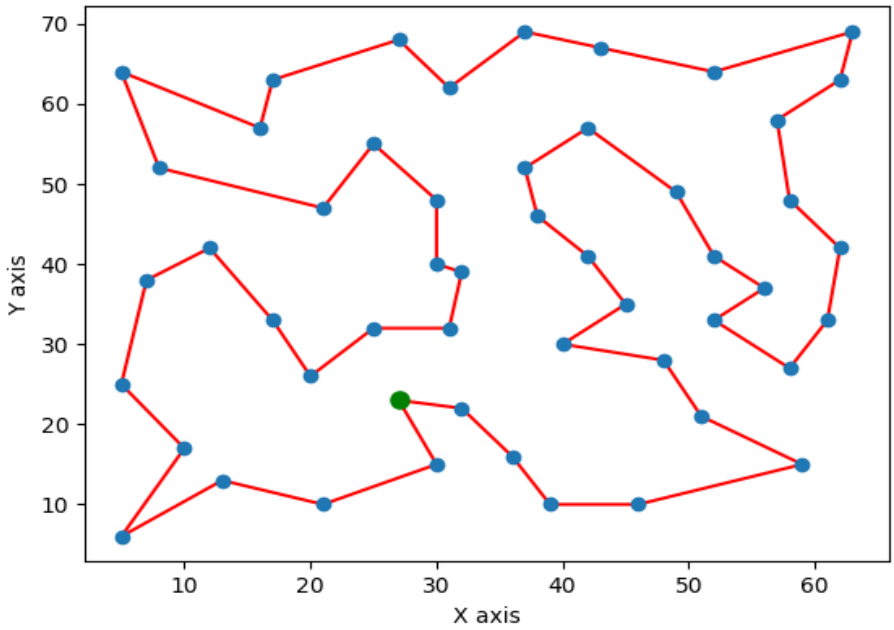


Fig. 2.25: Approximate solution of eil51 instance when using RSSCX as crossover. The total distance is 430.

2.5.2.1.2 Asymmetric TSP Similar to the results of Section 2.5.2.1.1, TSP instances were run and solved using each of the crossovers multiple times. Figure 2.26 shows solution quality comparison among crossovers when solving asymmetric TSP.

One observation in Figure 2.26 is that RSSCX and RSBCSCX outperform the other constructive crossovers (SCX, ESCX, and BCSCX) and the traditional operators (OX, CX, and PMX). Although RSSCX and RSBCSCX has about the same solution quality for some tested TSP instances, RSSCX outperforms RSBCSCX for most cases except in ry48p TSP instance. This outperformance of RSSCX in asymmetric TSP is expected because RSBC-SCX and its original BCSCX violate the asymmetric behavior when it looks for neighbor in right direction of the current city, while RSSCX and SCX respects this asymmetric property.

The performance graphs represented in Figures 2.27, 2.28, 2.29, 2.30, 2.31, 2.32, 2.33, and 2.34 show GA evolution convergence for solving the asymmetric TSP instances. As we can see in all of theses figures, RSBCSCX converged after 50 generations and even for

some TSP instances it converged after 30 generations, while other crossovers take longer to converge. We conclude that RSBCSCX takes less computational time to come up with a solution that is very close to optimal solution.

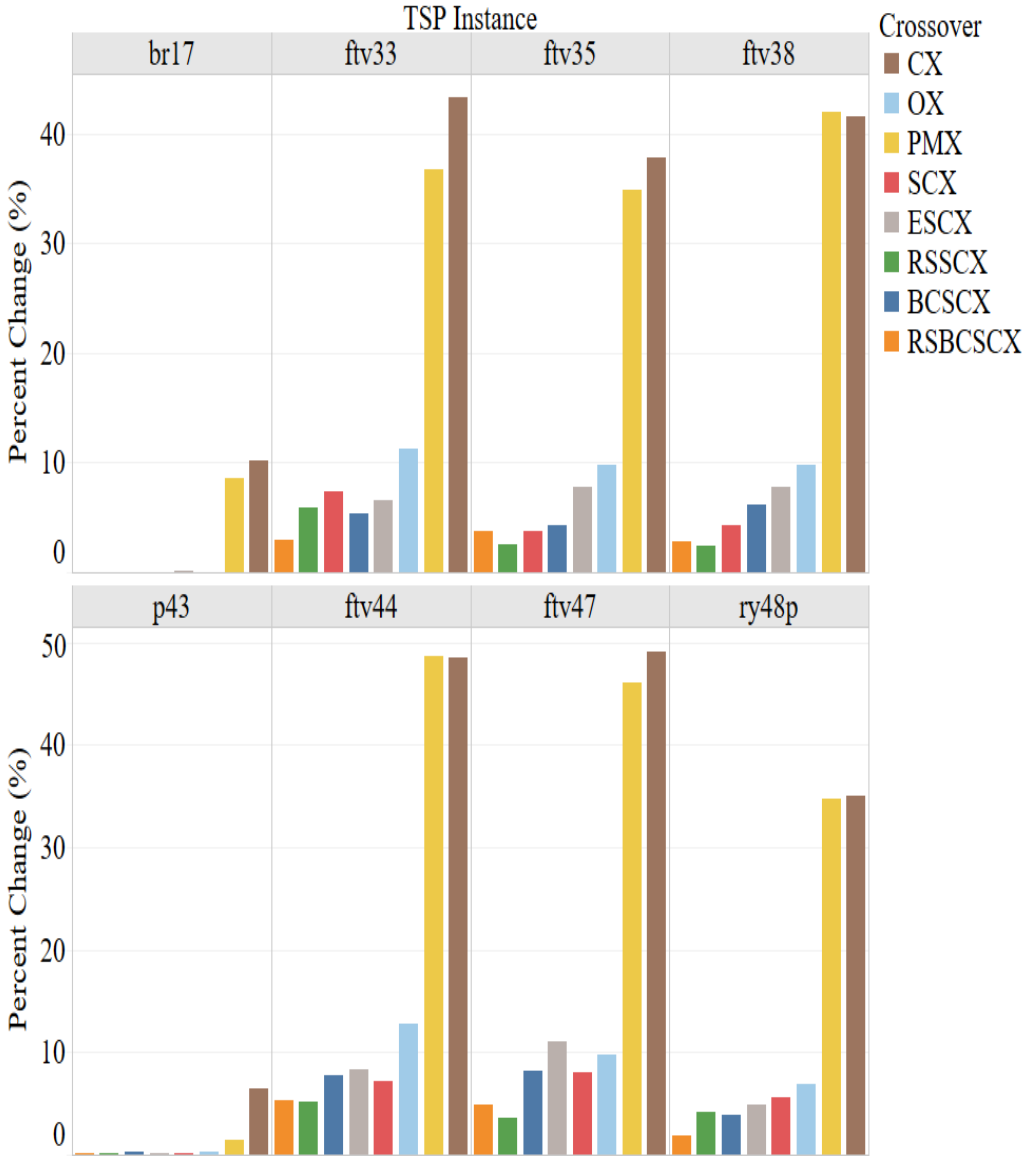


Fig. 2.26: Crossover comparison based on solution quality for asymmetric TSP.

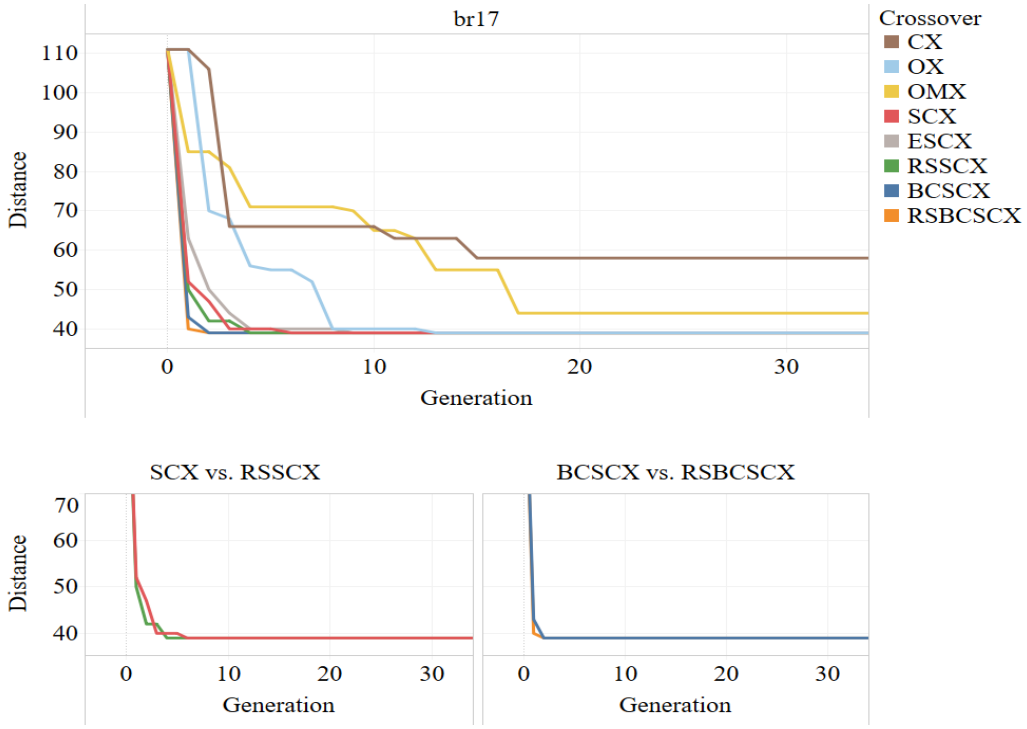


Fig. 2.27: Performance graph for br17 instance showing convergence over generations

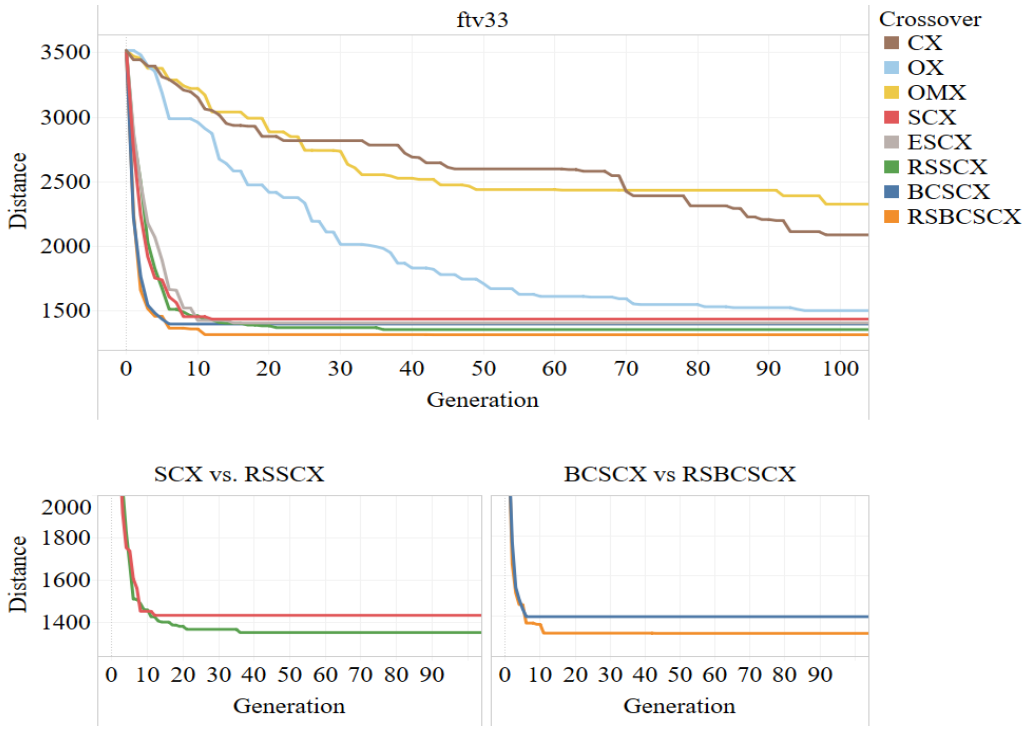


Fig. 2.28: Performance graph for ftv33 instance showing convergence over generations

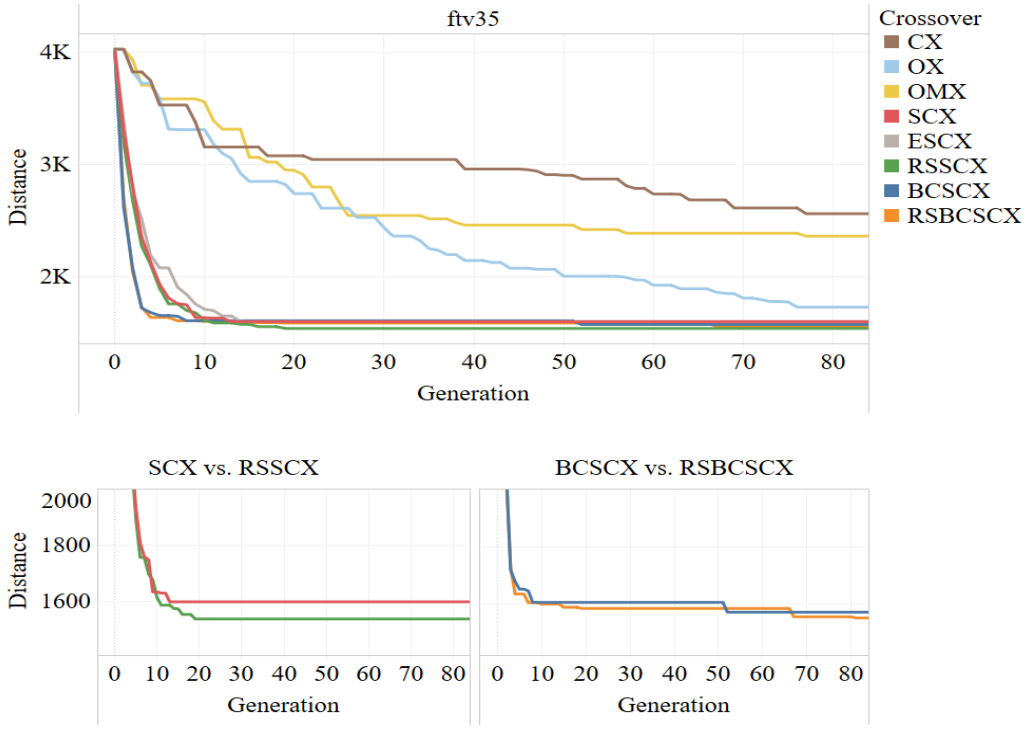


Fig. 2.29: Performance graph for ftv35 instance showing convergence over generations

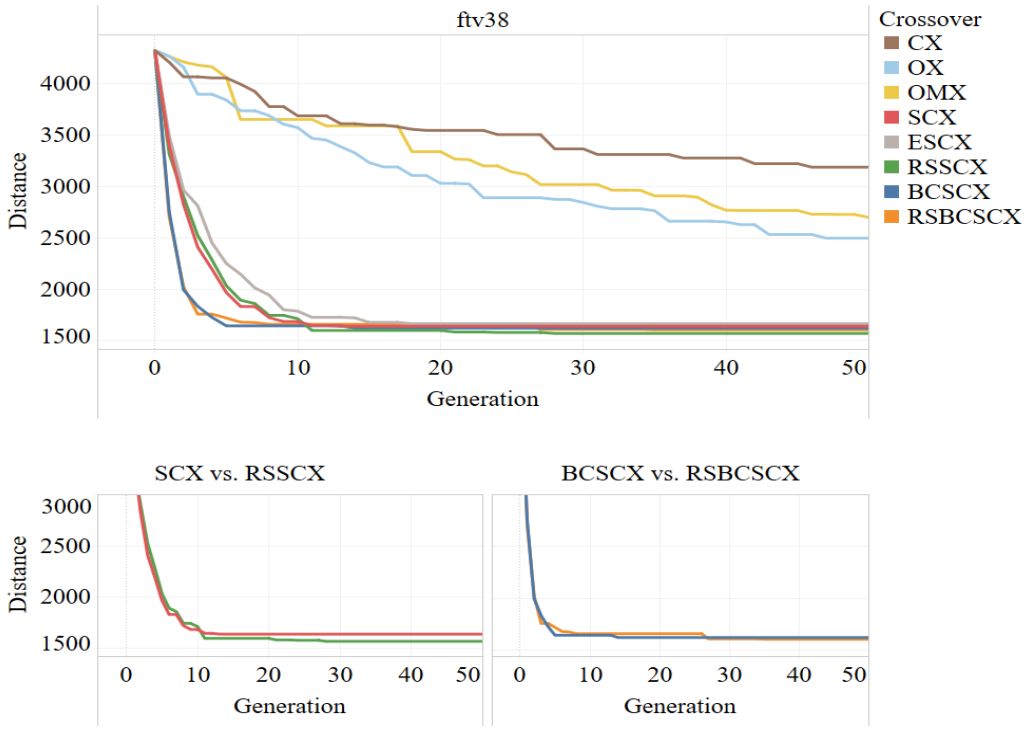


Fig. 2.30: Performance graph for ftv38 instance showing convergence over generations

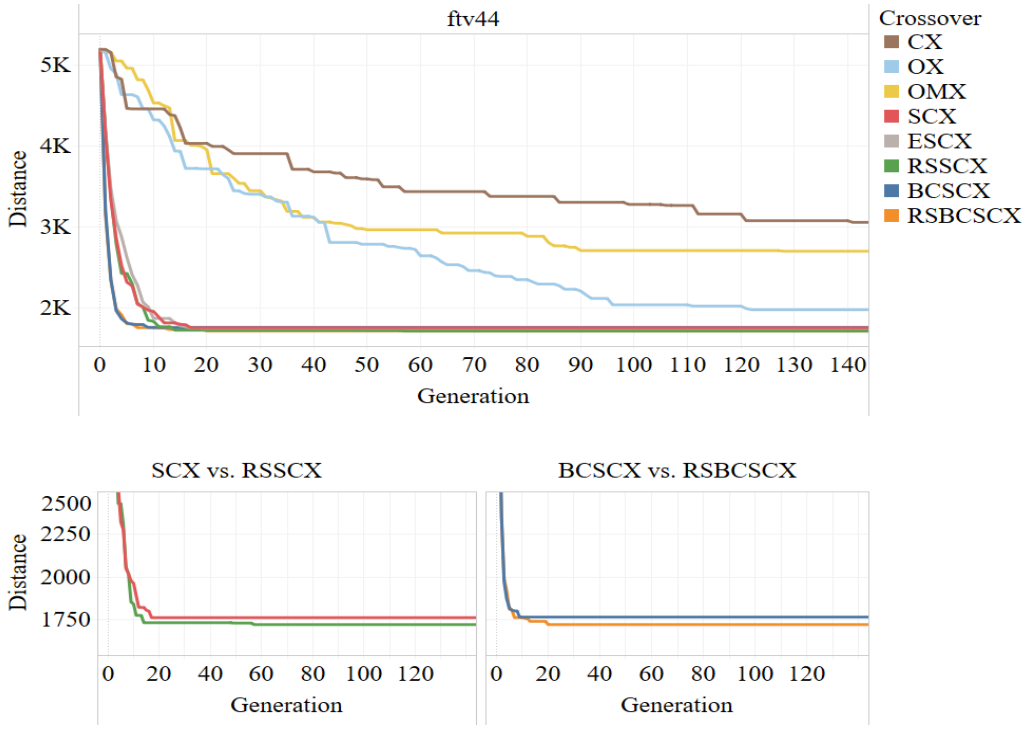


Fig. 2.31: Performance graph for ftv44 instance showing convergence over generations

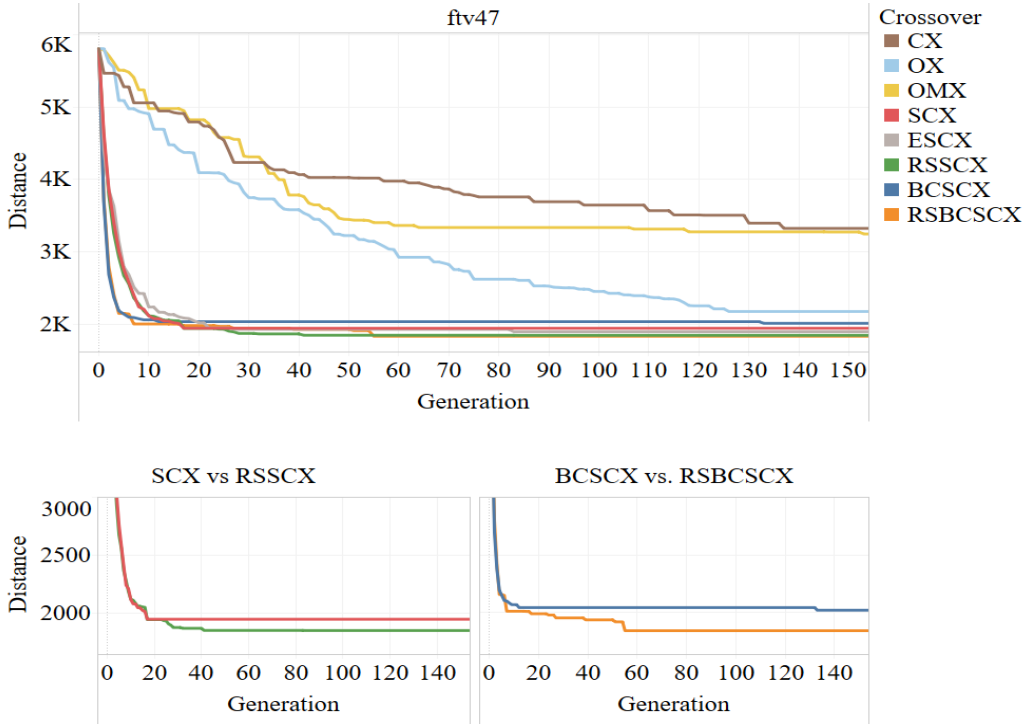


Fig. 2.32: Performance graph for ftv47 instance showing convergence over generations

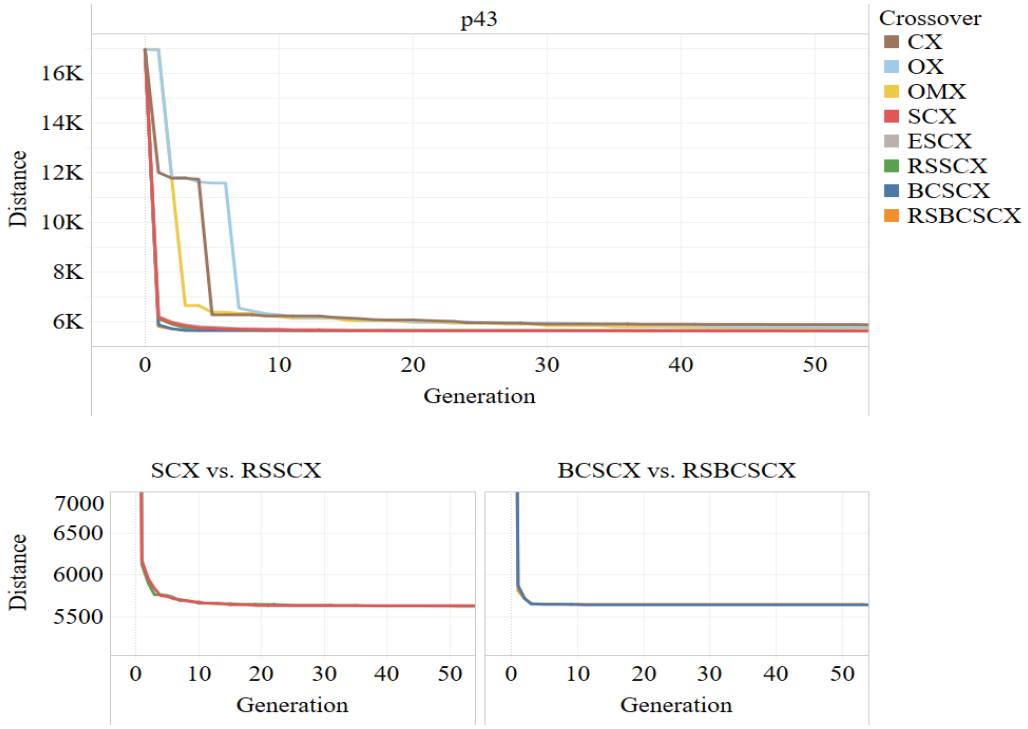


Fig. 2.33: Performance graph for p43 instance showing convergence over generations

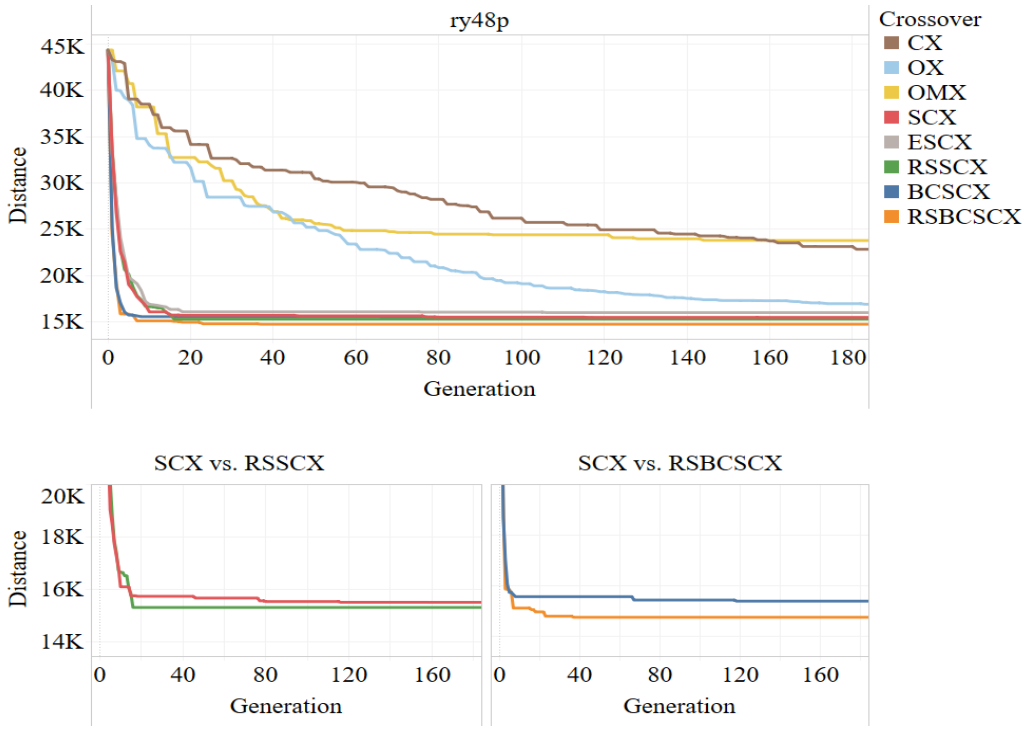


Fig. 2.34: Performance graph for ry48p instance showing convergence over generations

2.5.2.2 Computational Time

The following Figures 2.35 and 2.36 show the computational time that GA take to solve each TSP instance for both symmetric and asymmetric TSP respectively. This experiment is repeated 20 times for solving each instance with each of the crossovers and then we take the average time per instance per crossover. To get reliable time comparison among different crossovers, we set the number of generations and population size to 100.

There are three insights in these Figures 2.35 and 2.36. First, the traditional crossover operators (OX, CX, and PMX) take much less time for solving TSP compared to constructive operators (SCX, ESCX, RSSCXX, BCSCX, and RSBCSCX). That is due to the simplistic process that these crossovers follow when producing new offspring. Although the time efficiency is less, its solution quality is worse than constructive crossovers as shown in Figure 2.12 and Figure 2.26. Second, ESCX gives the worst computational time complexity among all operators. Third and the most important insight is that both our proposed operators RSSCX and RSBCSCX take about the same time in constructing offspring compared to the original SCX and BCSCX. Although RSSCX and RSBCSCX have same time complexity compared to the original methods, they yield much better solution quality as it is shown in Figure 2.12 and Figure 2.26.

2.5.2.3 Local Search Operator

To see the effectiveness of applying a local search operator on GA, we apply a non-uniform local search operator on symmetric and asymmetric TSP. The probability of applying this operator is set to 0.5 in our experiments. The Figures 2.37 and 2.38 show the comparison between using non-uniform local search and no search operator. The results show that applying a non-uniform local search operator give better solution quality for both symmetric and asymmetric TSP instances. Furthermore, the time complexity of this operator is $O(1)$ and is applied to GA within a probability. Thus, this operator is considered inexpensive and practical to be applied to GA.

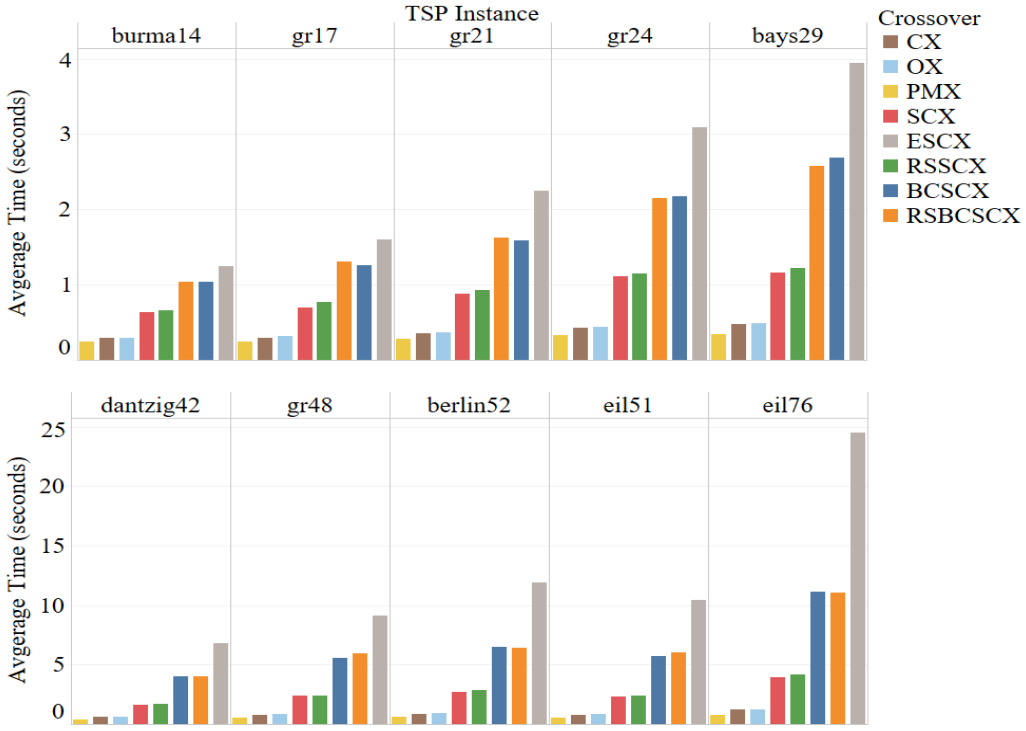


Fig. 2.35: Computational time comparison for symmetric TSP.

Fig. 2.36: Computational time comparison for asymmetric TSP.

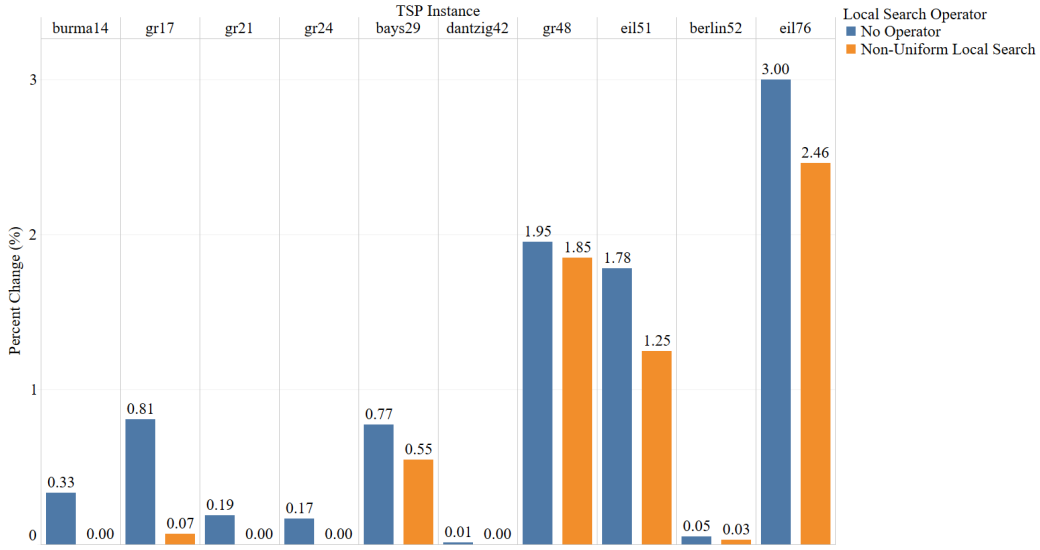


Fig. 2.37: Applying non uniform local search operator vs. no local search operator on GA for symmetric TSP.

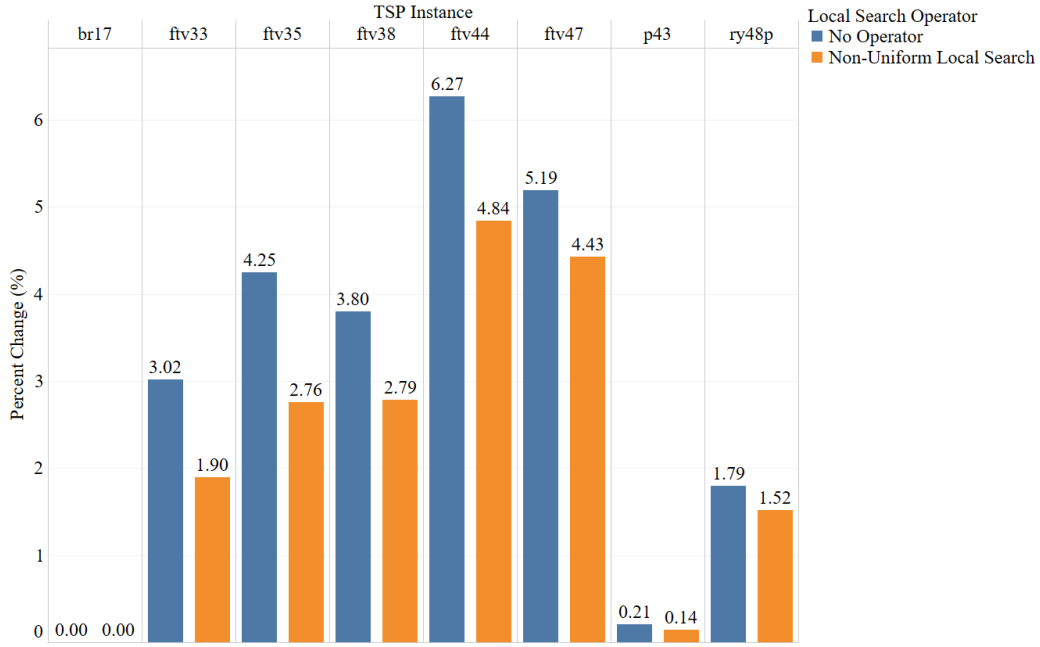


Fig. 2.38: Applying non uniform local search operator vs. no local search operator on GA for asymmetric TSP.

2.6 Conclusion and Future Work

In this paper, we propose a new modification to SCX and BCSCX crossovers to solve

both symmetric and asymmetric TSP. This modification requires the process start at random city when constructing offspring from parents in the genetic algorithm for solving both symmetric and asymmetric static TSP. This modification is called RSSCX and RSBCSCX respectively. Our proposed adjustment is tested against well-known TSP instances [18]. The experimental results show that our modification is superior to SCX, BCSCX, and other traditional crossovers in terms of solution quality. In addition, RSSCX and RSBCSCX have better convergence speed compared to other crossovers when it experimented with TSPLIB benchmarks. Moreover, computational time experiments show that our proposed crossover takes about the same time as its original crossovers (SCX and BCSCX) while it produces better solution quality. Although traditional crossover (OX, CX, and PMX) has better time complexity, it produces worse solution compared to constructive crossovers. Furthermore, we improve the GA efficiency by applying non-uniform local search operator. The experimental results indicate that applying this operator has a good impact on solution quality. Since this operator is inexpensive computationally (i.e. $O(1)$), then it is recommended as a practical operator to enhance the solution quality in GA.

For future study, one direction could be to apply the basic SCX and BCSCX along with our proposed modification to find a tour among destinations where the cost among these places reflects the traffic time in real world. Thus, the time between cities depends on the location in the tour.

CHAPTER 3

SOLVING TIME-DEPENDENT TSP USING GENETIC ALGORITHM

3.1 Abstract

The Time-Dependent Traveling Salesman Problem (TDTSP) is an interesting problem and has real impact on real-life applications such as a delivery system. In this problem, time among destinations fluctuates during the day due to traffic, weather, accidents, or other events. Thus it is important to recommend a tour that can save driver's time and resources. In this research, we propose a Multi-Population Genetic Algorithm (MGA) where each population has different crossovers. We compare the proposed MGA against Single-Population Genetic Algorithm (SGA) in terms of tour time solution quality. Our finding is that MGA outperforms SGA. Our method is tested against real-world traffic data [1] where there are 200 different instances with different numbers of destinations (i.e. 60 different instances of 10 destinations, 60 different instances of 20 destinations, 60 different instances of 30 destinations, and 20 different instances of 50 destinations). For all tested instances, MGA is superior on average by at least 10% (for instances with size less than 50) and 20% (for instances of size 50) better tour time solution compared to SGA with OX and SGA with PMX operators, and at least 4% better tour time compared to SGA with SCX operator.

3.2 Problem Description and Benchmark

3.2.1 Time-Dependent TSP Overview

The static TSP begins with a set of N cities and an $N \times N$ cost matrix, where each entry in the matrix represents the distance to travel from $City_i$ to $City_j$. The goal is to find the minimum distance of a tour in which a salesman visits each city exactly once and returns to the starting city. On the other hand, in time-dependent TSP, the cost (i.e. time)

between each two cities varies and is dependent on the time of the day. The following Equation 3.1 shows the time duration it takes to leave $City_i$ at time t and heading to $City_j$.

$$Cost_{i,j,t} = TravelTime(City_i, City_j, t) \tag{3.1}$$

Thus, the goal of TDTSP is to minimize the tour time by visiting each city exactly once and returning to the same starting city at the end as in Equation 3.2.

$$f = \min \sum_{i=1}^{n-1} Cost_{i,i+1,t} + Cost_{n,1,\bar{t}} \tag{3.2}$$

where t is the time of leaving $City_i$ and is updated by Equation 3.3

$$t_{i+1} = t_i + Cost_{i,j,t_i} \tag{3.3}$$

3.2.2 Time-Dependent TSP Benchmark

The benchmark provided by [1] is used in this research. This benchmark is created from real traffic data collected within 6 years in Lyon city in France. They consider 255 random delivery locations in Lyon city. The travel time step is 6 minutes from 6:00 AM to 12:30 PM. Thus, in total, there are 65 time steps. The travel time function is represented as a stepwise function as is shown in Figure 3.1.

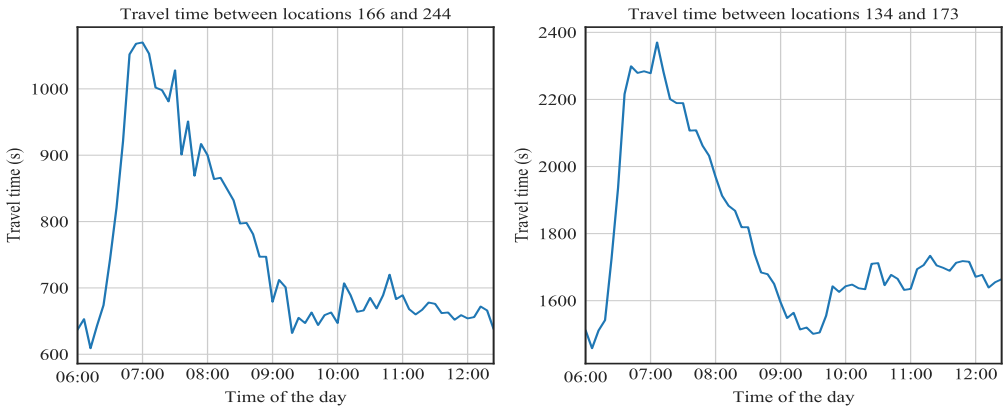


Fig. 3.1: Travel time function from 6:00 AM to 12:30 PM

In this benchmark, a service time (duration) at a delivery location is randomly chosen from 1 minute to 5 minutes. There are different problem sizes (i.e. 10, 20, 30, and 50). Each problem size has 60 different instances (except instances of size 50, which has 20 different instances).

3.3 Literature Review

In 1992, Malandraki and et al. [40] introduced TDTSP and general Time-Dependent Vehicle Routing Problem (TDVRP). The difference between TDVRP and TDTSP is that there are $N > 1$ vehicles that need to be routed in TDVRP whereas there is only one vehicle in TDTSP. They showed that some characteristics of static TSP do not hold for TDTSP. These characteristics are (1) an optimal tour cannot intersect itself and (2) the convex hull property. Thus, the time dependent version is more complex than the static version of TSP and some static TSP solution models are invalid for TDTSP. Also, they suggested some heuristics to solve this problem. The heuristics are nearest neighbor heuristic and cutting plane heuristic algorithm. They consider three different nearest neighbor methods: NN1, NN2, and NNR. NN1 works by starting at the depot and greedily adds the next nearest customer. NN2 repeats NN1 method $N - 1$ times, where each time it considers $Customer_i$ as next second visit and builds the rest of the tour as in NN1. NNR considers the next nearest customer depending on a predefined probability for first, second, and third best choices. They compared these heuristics using a randomly generated problem with different size (i.e. 10 to 25 customers) and found out there is no dominated heuristic among all tested methods that can solve TDTSP. Our solution will use more general heuristics (meta-heuristics) to give an approximate solution for TDTSP. Although [40] defined and described the problem of time dependency in TSP very well and in a clear way, the heuristics they used are extremely simple and cannot produce a reliable and near optimal solution for TDTSP.

Testa and et al. [38] studied the effect of different combinations of genetic algorithm operators on solving TDTSP. In other words, they tried to find out which best GA operator combinations were able to produce a high solution quality for TDTSP. They considered

eight operators in addition to adaptive operator probability and population re-initialization mechanism. The adaptive operator probability is a method to weight each operator by observing the importance of an operator that contributes to an improved solution. Three crossovers have been studied: recombination edge crossover, merge crossover, and cycle crossover. Recombination edge crossover considers the edges of each city in the parents to produce offspring. Merge crossover keeps the precedence of each city, for example, if $City_i$ comes before $City_j$ in both parents then $City_i$ should be visited before $City_j$ in the offspring. Cycle crossover keeps the positioning of the cities represented in the parents. Two mutation operators have been used: scramble sublist mutation and uniform order mutation. Scramble sublist mutation operates by randomly shuffling a consecutive sublist of cities. Uniform order mutation chooses two cities randomly and swaps them. They used two local search methods: uniform local search and non-uniform local search operator. Uniform local search chooses a consecutive sublist then permutes all possible orders of the cities in this sublist in a way to find out which best permutation can give lowest tour time. Non-uniform local search operator differs from uniform local search by choosing N random cities rather than a consecutive sublist of cities. They found that edge recombination crossover and cycle crossover are important to produce a high quality solution with the presence of one of the mutations. The worst combination is when no mutation and no local search operator are present in GA. The re-initialization mechanism plays a significant role (about 85% of all operator combinations) in solving TDTSP. They built their experiments based on randomly generated problems of up to 50 cities, while our approach is tested on real traffic data.

Zheng-yu and et al. [41] proposed an improved GA that can solve TDVRP, where the first optimization goal is to minimize the number of vehicles and the second goal is to minimize total schedule time. They concluded that using a mixed initial population with local search operator can increase the speed of GA and result in high quality solutions. In their implementation, the mixed initial population consists of 1/4 population which is randomly generated, 1/4 population using nearest neighbor, 1/4 population using Solomon insertion, and 1/4 population using IMPART algorithm. They used two point crossover where this

crossover can produce infeasible solutions (i.e. some cities get repeated), and consequently, the infeasible solutions need to be fixed. As a result, two point crossover increases the computational complexity. Our method does not produce any infeasible solutions so there is no need to fix any solution. Moreover, [41] randomly introduced traffic to static VRP with four predefined congestion probabilities which could lead to unrealistic data.

Ant Colony Optimization (ACO) [42] could be used as a meta-heuristic to solve TDTSP. Hitoshi and Ochiai [43] proposed a new method based on Min-Max Ant System (MMAS) that outperforms the conventional MMAS in term of search rate. Thus, their new method is faster by 2.6 to 3.4 times than conventional MMAS to produce an approximate solution for TDTSP. In their research, they suggest a way to convert a static TSP benchmark [18] by introducing a travel time change function. They considered five benchmarks from 51 to 300 cities where time interval is 5 units for small size benchmark (less than 200 cities) and 300 units for a larger problem size. However, our method uses real traffic data and a much larger number of benchmarks to validate its effectiveness.

Mavrovouniotis and et al. [44] made a comparison between multi-colony ACO and single-colony ACO for solving TDTSP. Multi-colony ACO is divided into two types: homogeneous (i.e. all colonies share same behavior) and heterogeneous (i.e. each colony has different behavior). They reported that multi-colony ACO is superior to the single-colony approach. The reason for this behavior is that using the multi-colony approach helps to escape local stationary optimum when a change in the environment happens. Also, they concluded that migration process is an important factor to communicate knowledge among colonies to come up with a better solution for TDTSP. Their proposed model used benchmarks that have been generated in such a way as to keep same static optimal solution value when it is converted to a dynamic benchmark. Thus, they immolate a realistic dynamic environment. Our approach uses another type of meta-heuristic (GA) and is tested on real urban traffic data.

Other heuristic algorithms such as Tabu search [45], simulating annealing [46] and other heuristics [47] [48] [49] have been introduced to solve TDTSP and TDVRP. Our solution

model focuses on GA and its operators to provide an efficient solution for TDTSP.

3.4 Methods

This section provides detailed information about the single-population genetic algorithm and multi-population genetic algorithm that we use to solve TDTSP.

3.4.1 Single-Population Genetic Algorithm

We use Algorithm 1 as single-population genetic algorithm. No local search operator (i.e. optimization operator) is used in this version of the algorithm. We consider three versions of this algorithm, each one has different crossover:

- SGA-OX: OX (refer to Section 2.4.1.1 for details) is used as a crossover.
- SGA-PMX: PMX (refer to Section 2.4.1.3 for details) is used as a crossover.
- SGA-SCX: SCX (refer to Section 2.4.1.4 for details) is used as a crossover.

Single-population means there is only one population that evolves using predefined selection, crossover, and mutation operators. The population finally converges to an approximate TDTSP solution. The initial population is randomly initialized. Tournament selection (refer to Section 2.4 for more details) is used to select parents for the next generation. Finally, swap mutation as shown in Figure 3.2 is used to exploit the search space by modifying offspring.

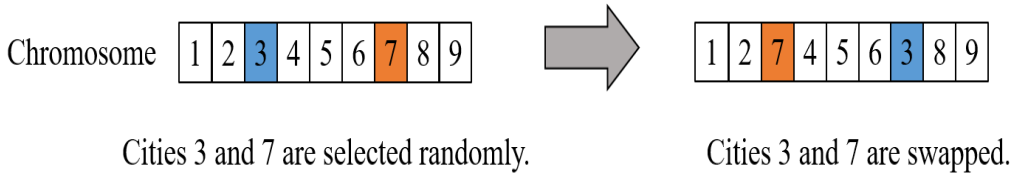


Fig. 3.2: Swap mutation operator.

3.4.2 Multi-Population Genetic Algorithm

Algorithm 3 shows our proposed Multi-population Genetic Algorithm (MGA).

Algorithm 3: Multi Population Genetic Algorithm

Data: TSP

Result: BestRoute

```
for g ← 1 to MaxGens do
    foreach Population ∈ Populations do
        if g == 1 then
            Population ← InitializePopulation(TSP)
        else
            ReinitializePopulation(Population − MigratedIndividuals)
        end
        Evaluate(Population)
        repeat
            Offspring ← Select(Population, PopulationSize − ElitistIndividualsSize)
            Crossover(Offspring)
            Mutate(Offspring)
            Evaluate(Offspring)
            Population ← PopulationElitistIndividuals + Offspring
            Update(PopulationElitistIndividuals)
        until Converged(Population)
    end
    Migrate(Populations)
end
BestRoute ← GetFittest(Populations)
```

The whole population is randomly initialized the first time. After the migration takes place, each population re-initializes its individuals so that the migrated individuals are unchanged. Re-initializing a population after convergence is an important process to give

a possibility to search in different areas in the search space. It allows a population to evolve and converge to a near optimal solution. The migration step is done by moving elitist individuals (e.g. best 10% of a population) from $population_i$ to $population_{i+1}$ in a circular manner. The migration happens after all populations converge. We assume that a population converges when there is no improvement over the best individual for three consecutive generations. In our implementation, there are three populations. Each population has a different crossover operator (i.e. $population_1$ has OX, $population_2$ has SCX, and $population_3$ has PMX).

The next section provides a performance comparison between SGA and MGA.

3.5 Experiments and Results

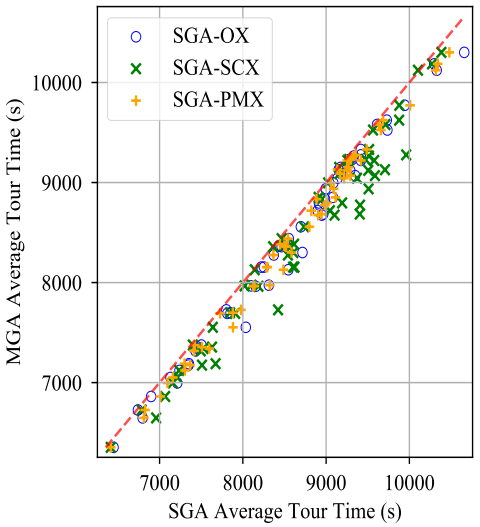
3.5.1 Experiment Setup

The objective of this experimental study is to investigate the performance of MGA compared to SGA when solving TDTSP instances in terms of solution quality on real traffic data [1].. The algorithms are coded in Python 3. For all experiments, we used random population initialization. We used tournament selection with tournament size set to 2. Swap mutation is applied in this experimental study.

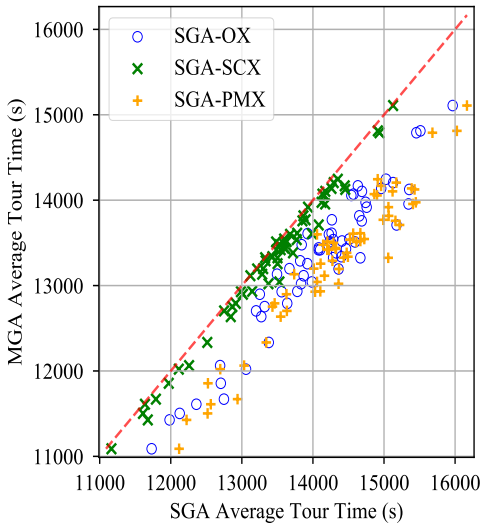
For this experiment, the guidelines of De Jong and Spears [39] have been followed, which recommend starting with a relatively high P_c and relatively low P_m, and population size is selected approximately 10 times larger than the number of cities in a problem. The maximum number of generations is chosen as stopping criteria for both SGA and MGA and it is set earlier in the program.

3.5.2 Experiment Results

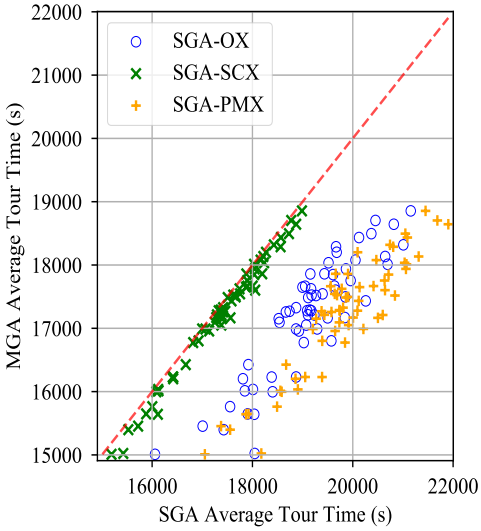
Figure 3.3 shows the solution average when solving TDTSP instances using SGA and MGA. In this figure, we solved each instance of the 200 instances (i.e. 60 instances of size 10, 60 instances of size 20, 60 instances of size 30, and 20 instances of size 50) 20 times using each algorithm and then the average is taken. There are two interesting observations. First,



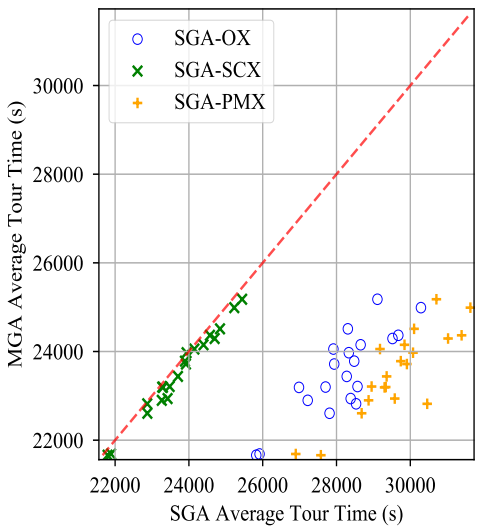
(a) 60 instances of size 10



(b) 60 instances of size 20



(c) 60 instances of size 30



(d) 20 instances of size 50

Fig. 3.3: SGA tour time average vs. MGA tour time average for different TDTDP instances.

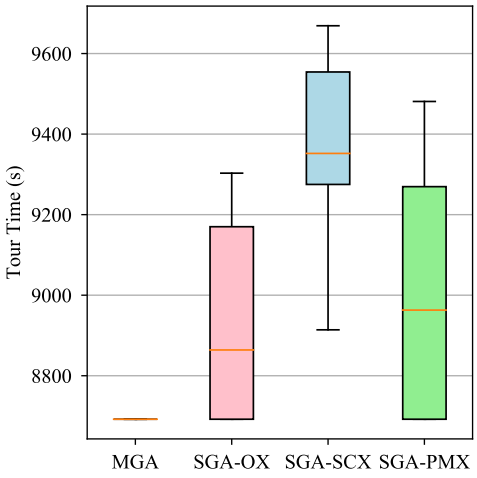
it is clear that MGA outperforms all types of SGA. Because of the current best solution to TDTSP may change over time due to the congestion among cities, the re-initialization part of the populations' solutions helps to evolve into new local optimal solution. Moreover, MGA has the advantages of searching the search space in three different ways (OX, SCX, and PMX). Second, one can observe that SGA with SCX operator is better than the other SGA types. This is due to the greedy behavior of SCX, which helps the population to converge to a local optimum solution faster and better than other crossovers (OX, PMX). Thus, combining the three crossover operators into one multi-population evolutionary algorithm is superior to the SGA with each of the crossovers.

Figure 3.4 shows the solution distribution of some instances in different sizes. Again the tour time solutions obtained by MGA are much better than each of SGAs. This figure shows the pattern of TDTSP solutions using each method. If we look at the centrality and the spread of the solutions, it is clear that MGA outperforms other methods. Another interesting observation is that SGA-SCX produces better results compared to other SGA as the instance size gets bigger, but it is still beaten by MGA method as is shown in Figures 3.4(b) 3.4(c) 3.4(d). Thus, in general, MGA outperforms all tested SGA methods for different problem sizes.

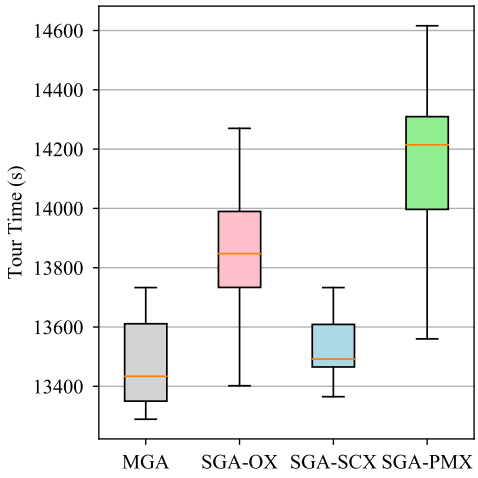
Finally, to measure the difference between SGA and MGA, we calculate the percentage difference between the average solutions of different instances and for all sizes (i.e. 10, 20, 30, and 50) using the following Equation 3.4.

$$PercentageDifference = \frac{|MGA_{solution} - SGA_{solution}|}{(MGA_{solution} + SGA_{solution}) \div 2} \times 100 \tag{3.4}$$

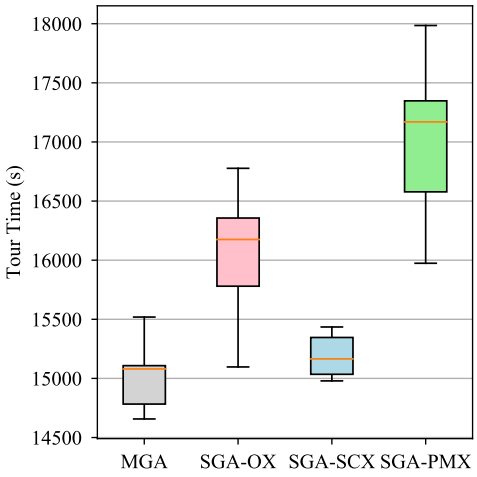
The percentage difference is shown in Figure 3.5. If we compare MGA against SGA-OX and SGA-PMX, the percentage difference is at least 10% for a problem size less than 50 and 20% when the problem size is 50. Also, if we compare SGA-SCX to MGA, we see that this crossover performs better than the other SGA methods and the lowest percentage difference is about 4% when problem size is 50. The difference is more than 5% for a problem size of less than 50.



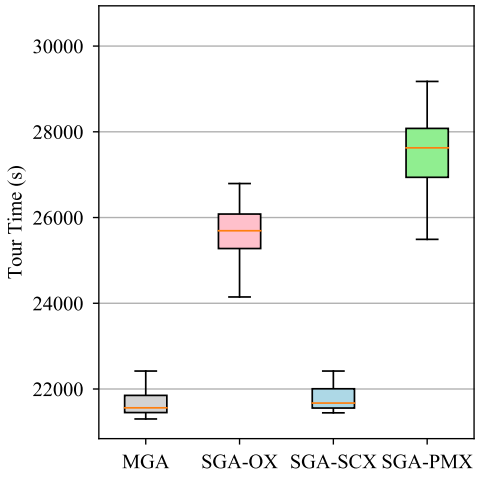
(a) Size 10 problem, instance $id =$ '$inst\_10\_5$'



(b) Size 20 problem, instance $id =$ '$inst\_20\_5$'

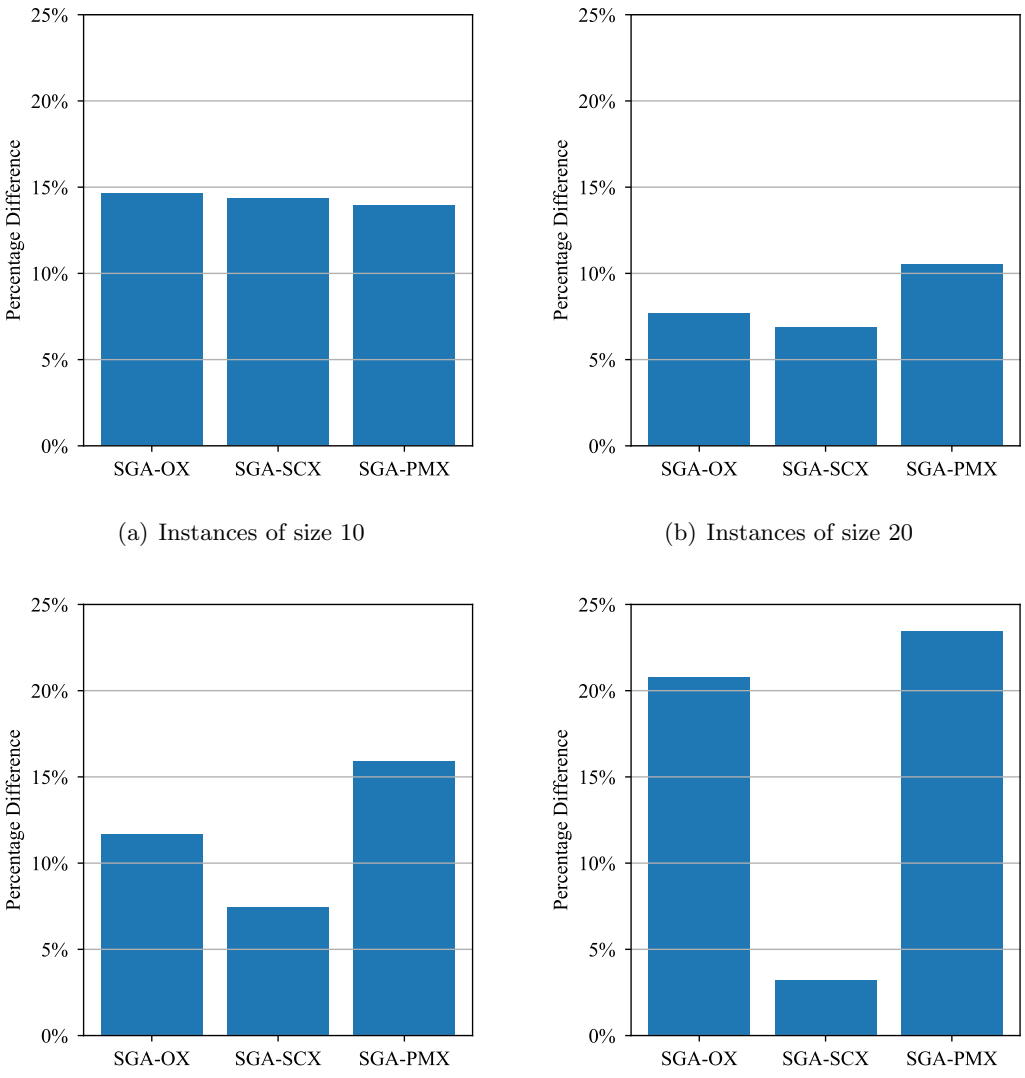


(c) Size 30 problem, instance $id =$ '$inst\_30\_5$'



(d) Size 50 problem, instance $id =$ '$inst\_50\_5$'

Fig. 3.4: Tour time solutions distribution over 20 runs for some instances.



(a) Instances of size 10

(b) Instances of size 20

(c) Instances of size 30

(d) Instances of size 50

Fig. 3.5: Percentage difference between SGA with (OX, SCX, and PMX) and MGA for solving TDTDP instances.

We use analysis of variance (ANOVA) [50] to test whether MGA and other SGAs are statistically significantly different. The significant difference means that we reject that there is no relationship between tested methods and a difference exists (null hypothesis). Table 3.1 shows the P-value and F-value. P-value is the probability of which we are confident to reject that there is no difference (i.e reject the null hypothesis). F-value is the ratio between methods solution mean square over within methods solution mean square. In other words, F-value measures solutions variability between different methods and solution variability within methods. As P-value < 0.05 (level of confidence) we conclude that at least one method is significantly different from other methods.

Table 3.1: One-way ANOVA for MGA, SGA-OX, SGA-SCX, and SGA-PMX.

	Problem Size 10	Problem Size 20	Problem Size 30	Problem Size 50
F value	12.449	343.439	1744.372	2462.421
P value	4.155e-08	2.960e-202	0.0	0.0

Now, to know which method(s) is significantly different from other methods, we perform Tukey's Honest Significant Difference (HSD) [51]. The HSD is a common post-hoc analysis to measure the significant difference among methods. The following Table 3.2 shows Tukey's HSD between MGA and other SGAs, where a mean difference is the difference between method 1 and method 2, lower and upper define the confident interval at which the two methods are different.

Table 3.2 confirms that MGA is significantly different from SGA-OX , SGA-SCX, and SGA-PMX for different problem size. Refer to [52] for more detailed explanation of ANOVA and Tukey's HSD test.

Table 3.2: Tukey HSD test results between MGA and other SGAs

	Method 1	Method 2	Mean Difference	Lower	Upper	Significant Difference
Size 10	MGA	SGA-OX	146.575	46.349	246.801	YES
P = 0.05	MGA	SGA-PMX	149.612	49.385	249.838	YES
	MGA	SGA-SCX	233.412	133.185	333.638	YES
Size 20	MGA	SGA-OX	774.395	679.596	869.194	YES
P = 0.05	MGA	SGA-PMX	1001.991	907.192	1096.789	YES
	MGA	SGA-SCX	137.598	42.799	232.396	YES
Size 30	MGA	SGA-OX	1936.997	1825.324	2048.669	YES
P = 0.05	MGA	SGA-PMX	2596.735	2485.062	2708.407	YES
	MGA	SGA-SCX	169.862	58.190	281.535	YES
Size 50	MGA	SGA-OX	4638.387	4424.652	4852.121	YES
P = 0.07	MGA	SGA-PMX	6096.842	5883.107	6310.576	YES
	MGA	SGA-SCX	213.824	0.090	427.559	YES

3.6 Conclusion and Future Work

In this paper, we compare two different genetic algorithms: single-population and multi-population genetic algorithms. Our proposed multi-population genetic algorithm has three populations, each population has a different crossover (i.e. $population_1$ uses OX, $population_2$ uses SCX, and $population_3$ uses PMX). Migration among elitist individuals is used to transfer knowledge among populations in a circular manner. The proposed algorithm is tested against single-population genetic algorithm with each of the crossovers (i.e. SGA-OX, SGA-PMX, SGA-SCX) using real-world traffic data [1]. Our finding is that MGA has superior performance compared to SGA in terms of tour time solution quality. The experiment is done on 200 different instances with different numbers of destinations (i.e. 60 instances for each problem size (10, 20, and 30), and 20 instances for the problem of size 50). Among single-population algorithms, only SGA-SCX has a better performance compared to other SGAs but it still beaten by MGA. The reason for the improvement of MGA in solving Time-Dependent TSP is the ability of MGA to explore the search space in three different manners, each manner corresponds to a different population, and that

ability increases the coverage of the search space of the problem. Also, the MGA uses the re-initialization process to randomly re-initialize a part of each population in each iteration to reset the evolved population again. As a result of the re-initialization process, a population can discover new local optimum that may be caused by a new change to the environment of TDTSP. In addition, the MGA uses a migration mechanism to transfer elitist individuals among populations to allow for faster convergence of each population and to use these migrated solutions in the process of evolving new local optimum next iteration.

For future studies, one possibility could be researching new ways to structure populations. Also, it would be interesting to investigate the performance of multi-population when each population has the same operators and multi-population algorithm uses different operators (e.g. crossover, mutation, or local search operators).

CHAPTER 4

CONCLUSIONS

The first part of this research deals with solving static TSP using the genetic algorithm. In this research, we propose a new modification to SCX and BCSCX crossovers to solve both symmetric and asymmetric TSP. This modification requires the process to start at random city when constructing offspring from parents in the genetic algorithm for solving both symmetric and asymmetric static TSP. This modification is called RSSCX and RSBCSCX respectively. Our proposed adjustment is tested against well-known TSP instances [18]. The experimental results show that our modification is superior to SCX, BCSCX, and other traditional crossovers in terms of solution quality. In addition, RSSCX and RSBCSCX have better convergence speed compared to other crossovers when it experimented with TSPLIB benchmarks. Moreover, computational time experiments show that our proposed crossover takes about the same time as its original crossovers (SCX and BCSCX) while it produces better solution quality. Although traditional crossover (OX, CX, and PMX) has better time complexity, it produces a worse solution compared to constructive crossovers. Furthermore, we improve the GA efficiency by applying non-uniform local search operator. The experimental results indicate that applying this operator has a good impact on solution quality. Since this operator is inexpensive computationally (i.e. $O(1)$), then it is recommended as a practical operator to enhance the solution quality in GA.

In the second part of this research, we study solving time-dependent TSP using the genetic algorithm. In this study, we compare two different genetic algorithms: single-population and multi-population genetic algorithms. Our proposed multi-population genetic algorithm has three populations, each population has a different crossover (i.e. $population_1$ uses OX, $population_2$ uses SCX, and $population_3$ uses PMX). Migration among elitist individuals is used to transfer knowledge among populations in a circular manner. The proposed algorithm is tested against single-population genetic algorithm with each of the crossovers

(i.e. SGA-OX, SGA-PMX, SGA-SCX) using real-world traffic data [1]. Our finding is that MGA has superior performance compared to SGA in terms of tour time solution quality. The experiment is done on 200 different instances with different numbers of destinations (i.e. 60 instances for each problem size (10, 20, and 30), and 20 instances for the problem of size 50). Among single-population algorithms, only SGA-SCX has a better performance compared to other SGAs but it is still beaten by MGA. The reason for the improvement of MGA in solving Time-Dependent TSP is the ability of MGA to explore the search space in three different manners (each manner corresponds to a different population) and that ability increases the coverage of the search space of the problem. Also, the MGA uses the re-initialization process to randomly re-initialize a part of each population in each iteration to reset the evolved population again. As a result of the re-initialization process, a population can discover new local optimum that may be caused by a new change to the environment of TDTSP. In addition, the MGA uses a migration mechanism to transfer elitist individuals among populations to allow for faster convergence of each population and to use these migrated solutions in the process of evolving new local optimum next iteration.

REFERENCES

[1] P. A. Melgarejo, P. Laborie, and C. Solnon, "A time-dependent no-overlap constraint: Application to urban delivery problems," in *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*. Springer, 2015, pp. 1–17.

[2] D. Schrank, B. Eisele, and T. Lomax, "2014 urban mobility report: Powered by inrix traffic data," Tech. Rep., 2015.

[3] D. Shinar, "Aggressive driving: the contribution of the drivers and the situation1," *Transportation Research Part F: traffic psychology and behaviour*, vol. 1, no. 2, pp. 137–160, 1998.

[4] R. G. Michael and S. J. David, "Computers and intractability: a guide to the theory of np-completeness," *WH Free. Co., San Fr*, pp. 90–91, 1979.

[5] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[6] R. Matai, S. Singh, and M. L. Mittal, "Traveling salesman problem: an overview of applications, formulations, and solution approaches," in *Traveling salesman problem, theory and applications*. InTech, 2010.

[7] J. Holland, "Adaption in natural and artificial systems," *Ann Arbor MI: The University of Michigan Press*, 1975.

[8] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi *et al.*, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[9] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 1, pp. 29–41, 1996.

[10] J. Kennedy and R. Eberhart, "Pso optimization," IEEE Service Center, Piscataway, NJ, pp. 1941–1948, 1995.

[11] F. Glover, "Tabu searchpart i," *ORSA Journal on computing*, vol. 1, no. 3, pp. 190–206, 1989.

[12] J.-Y. Potvin, "State-of-the-art surveythe traveling salesman problem: A neural network perspective," *ORSA Journal on Computing*, vol. 5, no. 4, pp. 328–348, 1993.

[13] B. Kylie, "Genetic algorithms and the traveling salesman problem," tech. rep., Department of Mathematics, Harvey Mudd College, Tech. Rep., 2000.

[14] Z. H. Ahmed, "Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator," *International Journal of Biometrics & Bioinformatics (IJBB)*, vol. 3, no. 6, p. 96, 2010.

[15] Wikipedia, "Genetic algorithm — Wikipedia, the free encyclopedia," 2017, [Online; accessed 30-May-2017]. [Online]. Available: https://en.wikipedia.org/wiki/Genetic_algorithm

[16] M. R. Noraini and J. Geraghty, "Genetic algorithm performance with different selection strategies in solving tsp," 2011.

[17] P. V. Paul, N. Moganarangan, S. S. Kumar, R. Raju, T. Vengattaraman, and P. Dhavachelvan, "Performance analyses over population seeding techniques of the permutation-coded genetic algorithm: An empirical study based on traveling salesman problems," *Applied Soft Computing*, vol. 32, pp. 383–402, 2015.

[18] G. Reinelt, "Tspliba traveling salesman problem library," *ORSA journal on computing*, vol. 3, no. 4, pp. 376–384, 1991.

[19] O. Abdoun and J. Abouchabaka, "A comparative study of adaptive crossover operators for genetic algorithms to resolve the traveling salesman problem," *arXiv preprint arXiv:1203.3097*, 2012.

[20] J. Grefenstette, R. Gopal, B. Rosmaita, and D. Van Gucht, "Genetic algorithms for the traveling salesman problem," in *Proceedings of the first International Conference on Genetic Algorithms and their Applications*, 1985, pp. 160–165.

[21] G. E. Liepins, M. R. Hilliard, M. Palmer, and M. Morrow, "Greedy genetics," in *Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA*. Hillsdale, NJ: L. Erlhaum Associates, 1987., 1987.

[22] J. Y. Suh and D. Van Gucht, "Incorporating heuristic information into genetic search," *JJ Grefen*, 1987.

[23] P. Jog, J. Y. Suh, and D. V. Gucht, "The effects of population size heuristic crossover and local improvement on a genetic algorithm for the traveling salesman problem," in *Proceedings of the 3rd International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers Inc., 1989, pp. 110–115.

[24] B. A. Julstrom, "Very greedy crossover in a genetic algorithm for the traveling salesman problem," in *Proceedings of the 1995 ACM symposium on Applied computing*. ACM, 1995, pp. 324–328.

[25] H. Ismkhan and K. Zamanifar, "Developing improved greedy crossover to solve symmetric traveling salesman problem," *arXiv preprint arXiv:1209.5339*, 2012.

[26] D. Whitley, T. Starkweather, and D. Shaner, *The traveling salesman and sequence scheduling: Quality solutions using genetic edge recombination*. Colorado State University, Department of Computer Science, 1991.

[27] N. J. Radcliffe and P. D. Surry, "Fitness variance of formae and performance prediction." in *FOGA*, vol. 3. Citeseer, 1994, pp. 51–72.

[28] L.-Y. Wang, J. Zhang, and H. Li, "An improved genetic algorithm for tsp," in *Machine Learning and Cybernetics, 2007 International Conference on*, vol. 2. IEEE, 2007, pp. 925–928.

[29] O. Abdoun, J. Abouchabaka, and C. Tajani, "Analyzing the performance of mutation operators to solve the travelling salesman problem," *arXiv preprint arXiv:1203.3099*, 2012.

[30] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," *Foundations of genetic algorithms*, vol. 1, pp. 69–93, 1991.

[31] T. Blickle and L. Thiele, "A comparison of selection schemes used in genetic algorithms," 1995.

[32] K. Sastry, D. E. Goldberg, and G. Kendall, "Genetic algorithms," in *Search methodologies*. Springer, 2014, pp. 93–117.

[33] A. E. Eiben, J. E. Smith *et al.*, *Introduction to evolutionary computing*. Springer, 2003, vol. 53.

[34] D. E. Goldberg, "Genetic algorithms in search, optimization, and machine learning, 1989," *Reading: Addison-Wesley*, 1989.

[35] D. E. Goldberg, R. Lingle *et al.*, "Alleles, loci, and the traveling salesman problem," in *Proceedings of an international conference on genetic algorithms and their applications*, vol. 154. Lawrence Erlbaum, Hillsdale, NJ, 1985, pp. 154–159.

[36] H. Bennaceur and E. Alanzi, "Genetic algorithm for the travelling salesman problem using enhanced sequential constructive crossover operator," *International Journal of Computer Science and Security (IJCSS)*, vol. 11, no. 3, p. 42, 2017.

[37] S. Kang, S.-S. Kim, J.-H. Won, and Y.-M. Kang, "Bidirectional constructive crossover for evolutionary approach to travelling salesman problem," in *IT Convergence and Security (ICITCS), 2015 5th International Conference on*. IEEE, 2015, pp. 1–4.

[38] L. J. Testa, A. C. Esterline, G. V. Dozier, and A. Homaifar, "A comparison of operators for solving time dependent traveling salesman problems using genetic algorithms," in *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc., 2000, pp. 995–1002.

[39] K. A. De Jong and W. M. Spears, "Using genetic algorithms to solve np-complete problems." in *ICGA*, 1989, pp. 124–132.

[40] C. Malandraki and M. S. Daskin, "Time dependent vehicle routing problems: formulations, properties and heuristic algorithms," *Transportation science*, vol. 26, no. 3, pp. 185–200, 1992.

[41] D. Zheng-yu, Y. Dong-yuan, and W. Shang, "An improved genetic algorithm for time dependent vehicle routing problem," in *Future Computer and Communication (ICFCC), 2010 2nd International Conference on*, vol. 1. IEEE, 2010, pp. V1–835.

[42] M. Derigo and T. Stutzle, "Ant colony optimization. the mit press," 2004.

[43] H. Kanoh and J. Ochiai, "Solving time-dependent traveling salesman problems using ant colony optimization based on predicted traffic," in *Distributed Computing and Artificial Intelligence*. Springer, 2012, pp. 25–32.

[44] M. Mavrovouniotis, S. Yang, and X. Yao, "Multi-colony ant algorithms for the dynamic travelling salesman problem," in *Computational Intelligence in Dynamic and Uncertain Environments (CIDUE), 2014 IEEE Symposium on*. IEEE, 2014, pp. 9–16.

[45] S. Ichoua, M. Gendreau, and J.-Y. Potvin, "Vehicle dispatching with time-dependent travel times," *European journal of operational research*, vol. 144, no. 2, pp. 379–396, 2003.

[46] J. Schneider, "The time-dependent traveling salesman problem," *Physica A: Statistical Mechanics and its Applications*, vol. 314, no. 1-4, pp. 151–155, 2002.

[47] W. Maden, R. Eglese, and D. Black, "Vehicle routing and scheduling with time-varying data: A case study," *Journal of the Operational Research Society*, vol. 61, no. 3, pp. 515–522, 2010.

[48] H. Hashimoto, M. Yagiura, and T. Ibaraki, "An iterated local search algorithm for the time-dependent vehicle routing problem with time windows," *Discrete Optimization*, vol. 5, no. 2, pp. 434–456, 2008.

[49] M. A. Figliozzi, "The time dependent vehicle routing problem with time windows: Benchmark problems, an efficient solution algorithm, and solution characteristics," *Transportation Research Part E: Logistics and Transportation Review*, vol. 48, no. 3, pp. 616–636, 2012.

[50] Wikipedia contributors, "Analysis of variance — Wikipedia, the free encyclopedia," 2018, [Online; accessed 13-April-2018]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Analysis_of_variance&oldid=835543684

[51] ——, "Tukey's range test — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=Tukey%27s_range_test&oldid=792559039, 2017, [Online; accessed 23-April-2018].

[52] Stan Brown, "Comparing more than two means: One-way anova," https://brownmath.com/stat/anova1.htm, 2017, [Online; accessed 17-April-2018].